UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Ingeniería de Telecomunicación especialidad Sistemas de Telecomunicación

PROYECTO DE FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN SIMULADOR DE RED KNX EIBSEC EN OMNeT++

Autor: Débora Iñiguez Sánchez
Tutor: Daniel Díaz Sánchez

Madrid, Junio de 2011

AGRADECIMIENTOS

Llegado este momento, no podía olvidarme de dar las gracias a las personas que me han acompañado en este largo viaje por la universidad y que han sabido entenderme en todo momento.

En primer lugar quiero agradecer a mis padres, Martin y Mercedes, el darme la oportunidad de haber estudiado una carrera, su apoyo incondicional y la motivación que me han brindado para poder seguir adelante en situaciones que me han hecho dudar. Gracias por vuestra confianza depositada en mí y la ayuda que me habéis dado que han hecho que hoy pueda estar aquí.

A mis hermanas, Sara y Nazaret, les doy las gracias por sus consejos, palabras de ánimo y sus risas me han hecho más llevadero momentos de tensión y agotamiento.

A alguien especial que ha estado ahí conmigo desde el primer momento que comenzó esta etapa final. Gracias Salva por demostrarme tu apoyo incondicional, darme fuerzas y regalarme una sonrisa en los momentos más difíciles.

Quiero agradecer a mi tutor del proyecto, Daniel Díaz, su interés, tiempo, esfuerzo, apoyo y dedicación en el proyecto, pero sobre todo las palabras y consejos que me has dado para superar esto y llegar hasta el final.

A mis compañeras de beca a día de hoy son amigas, Lorena y Victoria, muchas gracias por confiar en mí en todo momento, estar a mi lado en las alegrías y las tristezas y por lo buenos momentos que hemos compartido juntas y espero seguir compartiendo.

A mis amigos ajenos a la universidad que han sabido entenderme en todo momento y darme las palabras de ánimo que necesitaba.

A todos aquellos compañeros de la universidad que nunca han dudado en ayudarme.

RESUMEN

Los sistemas de automatización de edificios gestionan aplicaciones como el control de calefacción, aire acondicionado, iluminación e intensidad. KNX/EIB es un estándar a nivel internacional para el control de viviendas y edificios. EIBSec es una extensión de este estándar que soporta comunicaciones seguras, gestión de claves, integridad de datos, confidencialidad así como autenticación. Estos conceptos son necesarios para aplicación críticas de seguridad como el control de acceso y sistemas de alarmas de seguridad.

En este proyecto, primeramente se introducen los conceptos del protocolo KNX/EIB. Una vez revisados estos conceptos se introduce la funcionalidad de la extensión de seguridad en una red KNX/EIB. El desarrollo de una red KNX/EIB y su extensión EIBSec son analizados y comparados a través de su simulación.

Se ha empleado una herramienta de simulación llamada OMNeT++. Es un simulador modular con una arquitectura y código libres. Gracias a su potente soporte grafico permite al usuario seguir la fácilmente la simulación. Dispone de unas librerías básicas que ayudan a recolectar los datos durante la simulación para su posterior procesado.

ABSTRACT

Building automation systems deal with the control of heating, air conditioning, lighting and dimming. KNX/EIB is an international standard in home and building automation control. EIBSec is an extension to this standard and supports secure communications, key management, data integrity, confidentiality and authentication. These concepts are necessary to implement critical application like access control and security alarm systems.

In this project first the concepts of KNX/EIB will be introduced. Once reviewed these concepts, security extension functionality will be introduced. The performance of KNX/EIB and its extension EIBSec are analyzed and compared by simulation.

The OMNeT++ simulation has been used. The simulation framework OMNeT++ is a modular architecture and open source. Due to powerful graphical supports allow understand the simulation easily. OMNeT++ has some basic libraries that help collect data while the simulation. After that, data collection can be processed.

ÍNDICE GENERAL

INDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1: INTRODUCCIÓN

1.1 MOTIVACIÓN DEL PROYECTO

Inicialmente surge *European Installation Bus (EIB),* con el propósito de introducir en el Mercado un sistema que unifique la gestión de edificios. Con este planteamiento inicial se crea un consorcio europeo llamado *European Installation Bus Association (EIBA),* fundado en 1990 por más de setenta compañías (ABB, Siemens…) **[1].** El objetivo de esta asociación fue la unión de varios estándares: *European Home System (EHS), Batibus y EIB.* Con esta unión lo que se pretendía era conseguir la compatibilidad de productos entre ellos. EIBA es la creadora y propietaria de la tecnología KNX cuyo estándar ISO/IEC 14543 fue aprobado mas tarde a nivel internacional. Este estándar es el único estándar aprobado a nivel internacional para el control de viviendas y edificios **[2].**

EIB o KNX tiene un problema de seguridad dado que los mensajes se envían en claro, sin ningún tipo de protección. El Sistema de Automatización de Grupo de la Universidad Tecnológica de Viena, pionero de la investigación en el sistema de automatización de edificios, se centró sobre todo en las extensiones de seguridad para el protocolo de domótica KNX EIB **[3]**. En concreto se ha desarrollado una extensión de seguridad del protocolo EIB llamado *Security European Installation Bus (EIBSec)*, que debe ser evaluado y comparado en cuanto a su rendimiento.

En este proyecto se va a verificar el comportamiento y rendimiento de ambos casos. Aparte de esta verificación, nos aporta facilidad para realizar todo tipo de pruebas requeridas sobre el protocolo, sin limitación alguna en costes de dispositivos y tiempo empleado en instalación.

En reglas generales, hay distintos procedimientos para poder hacer dicha evaluación entre los que cabe destacar los procedimientos analíticos, uso de prototipos y simulaciones. En este caso, se ha optado por la simulación del protocolo EIB así como su extensión de seguridad.

La opción elegida ha sido la simulación debido a que nos permite hacer uso de cualquier dispositivo deseado, así como la simulación de todo tipo de redes. Por eficacia, nos concentramos en algunos aspectos concretos del sistema para obtener unos resultados clave como diferencias entre un modo seguro y no seguro. Por ello, hay ciertos aspectos del protocolo original que no se tienen en cuenta ya que no son clave para los resultados que queremos

obtener. Aunque tenemos un propósito concreto, tampoco se pueden omitir muchos parámetros y aspectos del comportamiento de EIB, de lo contrario no obtendríamos un comportamiento muy real de la red, siendo otro de nuestros objetivos a cumplir.

Existen múltiples detonantes que nos llevan a realizar una simulación como puede ser: visualización del comportamiento de la red teniendo en cuenta que por complejidad una red real no se puede construir, diferencias entre una red con funcionamiento no seguro como es EIB a un desarrollo a partir de esta con un funcionamiento en modo seguro, y la posibilidad de implementar nuevas mejoras futuras a partir de la base de este proyecto sin necesidad de hacer muchos cambios.

En cambio si la elección hubiese sido hacer uso de un prototipo con componentes reales, nos estaría limitando en el desarrollo ya que no sería posible utilizar muchos dispositivos. Este caso sería mejor utilizarlo para mejoras de diseño y verificación del comportamiento.

1.2 OBJETIVOS

En cuanto a los objetivos que se quieren conseguir con la implementación de este protocolo y su ampliación de seguridad son:

- Desarrollo e implementación del funcionamiento básico del protocolo KNX EIB, para conseguir la comunicación entre los distintos dispositivos EIB. Desarrollo de otras funcionalidades auxiliares para conseguir el comportamiento especificado.

- Implementación e inclusión de dispositivos que forman parte del estándar.

- Desarrollo de comportamiento del modelo de seguridad KNX a partir del funcionamiento no seguro EIB.

- Análisis y comparación de los aspectos más relevantes de ambas redes como carga de la red, número de veces que el bus está ocupado, número total de mensajes que se intercambian, numero de bytes que se envían y reciben, es decir, el comportamiento en ambos casos.

- Facilidad para realizar posibles en el futuro como puede ser nuevos dispositivos no implementados o nuevos desarrollos a nivel de protocolos.

- Realizar pruebas de distintas redes a las simuladas como por ejemplo una red extensa de gestión de un edificio. De esta manera previamente se puede observar el comportamiento de dicha red, asemejándose a la red real a construir.

1.3 CONTENIDO DE LA MEMORIA

El presente proyecto consta de una serie de capítulos dedicados a las distintas fases por las que ha pasado el proyecto:

- ***Capítulo 1: Introducción***. Marco de trabajo donde se ha realizado el proyecto comentando el entorno y la motivación que ha llevado al estudio y la realización del presente, así como los objetivos marcados inicialmente a tener en cuenta para el desarrollo del mismo.

- ***Capítulo 2: Estado del arte.*** Ofrece un resumen con los conceptos más importantes del protocolo KNX EIB. Además de esto, se ha incluido parte técnica del funcionamiento de dicho protocolo con seguridad habilitada. Análisis de los posibles dispositivos conectados a estas redes así como su instalación.

- ***Capítulo 3: Introducción a OMNeT++.*** Conceptos básicos y utilización del framework como herramienta de desarrollo del proyecto.

- ***Capítulo 4: Diseño del proyecto.*** Decisiones iniciales que se han llevado a cabo antes del desarrollo, así como el diseño inicial de los módulos que lo componen.

- ***Capítulo 5: Desarrollo de los módulos OMNeT++.*** Decisión final y descripción detallada de los distintos módulos desarrollados en OMNeT++ que componen el proyecto.

- ***Capítulo 6: Diseño y desarrollo de una aplicación.*** Pruebas realizadas para el análisis de resultados. Modelo de simulación implementado, criterios seguidos, propósito y organización de estas. Nos ayudan a verificar la funcionalidad del protocolo así como comparar los distintos modos de funcionamiento analizando los resultados.

- ***Capítulo 7: Historia del proyecto.*** Descripción de problemas, soluciones adoptadas para su resolución y replanteamientos de desarrollo. Seguimiento del desarrollo del proyecto.

- ***Capítulo 8: Conclusiones y trabajos futuros.*** Conclusiones de los objetivos realizados y cumplidos, así como ampliaciones y mejoras posibles a realizados en un futuro.

Capítulo 2: ESTADO DEL ARTE

2.1 INTRODUCCIÓN

Desde el punto de vista etimológico, la palabra domótica fue inventada en Francia y está formada por la contracción de "domus" que significa vivienda más "automática" **[4]**. Aunque la palabra indica que se refiera a vivienda automática, este campo no solo se refiera únicamente a la vivienda, sino a cualquier tipo de edificación. Además, la domótica va más allá de la mera automatización de un edificio, integrando el control del mismo con el uso que se hace de él.

Se pueden distinguir tres sectores distintos dependiendo del alcance de su aplicación **[5]:**

- Domótica, para el sector domestico.
- Inmótica, para el sector terciario.
- Urbótica, para las ciudades. En este caso se tratan temas como el control de la iluminación pública, gestión de semáforos, medios de pago, etc.

Para definir una vivienda automatizada hay que tener en cuenta dos puntos de vista: el del usuario y el técnico. Desde el punto de vista del usuario, una vivienda domótica es aquella que proporciona mayor calidad de vida a través de las nuevas tecnologías, ofreciendo una reducción del trabajo domestico, un aumento del bienestar y la seguridad de sus habitantes. Desde el punto de vista técnico, es aquella que se integran los distintos aparatos domésticos que tienen la capacidad de intercomunicarse entre ellos a través de un soporte de comunicaciones, de modo que puedan realizar tareas que hasta ahora se venían haciendo de manera manual.

La estructura general de la automatización de una instalación es la siguiente:

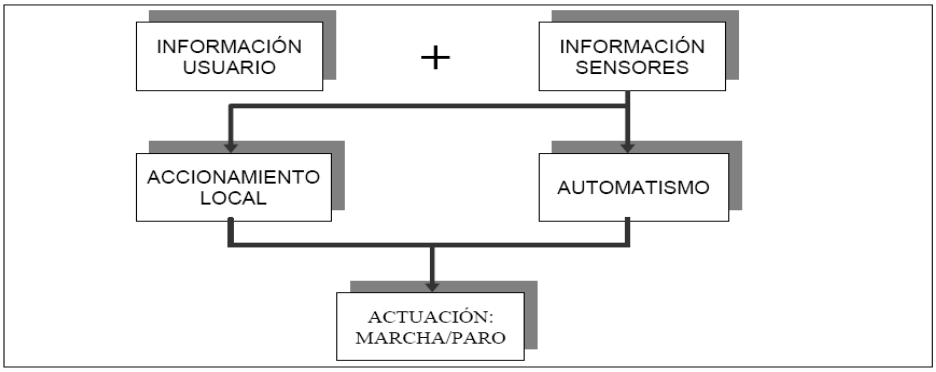


Figura 1 Estructura general de la automatización de una instalación

Hoy en día, existen diferentes sistemas domóticos que se pueden utilizar en función del tamaño de la edificación. Desde el punto de vista comercial, puede decirse que actualmente los sectores que precisan de estos sistemas son casas ya construidas, casas nuevas y grandes edificios. Cada uno de estos sectores utiliza una tecnología específica, adaptada a la necesidad del usuario final que son **[5]**:

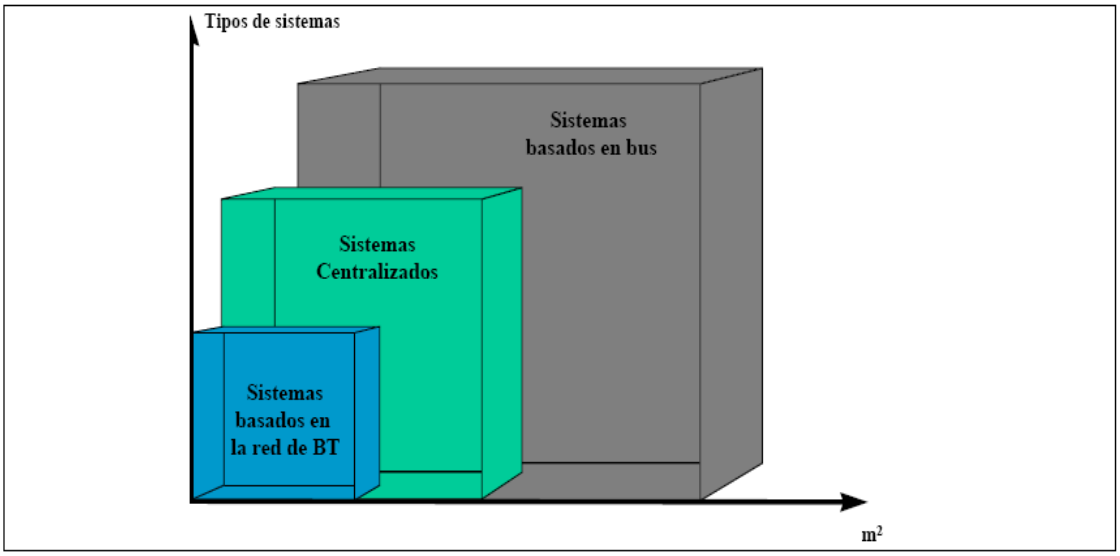


Figura 2. Diferentes sistemas empleados en las instalaciones domóticas

Los sistemas basados en la red de *Baja Tensión (BT)* son los de corrientes portadoras que tiene como soporte la propia red de alimentación de baja tensión de 220V, presente ya en la vivienda. Este tipo de tecnología se utiliza en casas ya construidas por el elevado costo, y a veces la imposibilidad, de hacer un precableado nuevo para el sistema domótico.

Si se trata de una casa nueva, dependiendo del tamaño y requisitos, los más apropiados son los *Sistemas Centralizados Comerciales (SCC).* Las gamas altas de SCC se emplean en viviendas nuevas de tamaño grande con

necesidades avanzadas, mientras que las de gama baja son para nuevas viviendas de pequeño tamaño sin grandes requerimientos. El uso de este tipo de sistemas cuenta con la ventaja de tener un coste muy reducido y no requiere ningún tipo de especialización para su instalación.

Por último, los *Sistemas basado en Bus* son los que se instalan en edificios debido a la complejidad de sus requerimientos y necesidades. A nivel europeo uno de los sistemas más importantes es EIB.

2.2 INTRODUCCIÓN AL EIB

El EIB surgió con la idea de introducir en el mercado un sistema unificado para la gestión de edificios, creados por el consorcio europeo EIBA, cuyas siglas significan European Installation Bus Association, en 1990 por más de setenta compañías (ABB, Siemens …) **[1]**

En la actualidad la asociación tiene más de cien miembros, existiendo unas veinte empresas que suministran productos, siendo las más importantes Siemens, ABB, Temper, Grasslin y Niessen. También existen miembros científicos que colaboran en el desarrollo de actividades de investigación y desarrollo, especialmente universidades y centros de investigación.

El EIBA (KNX Association) tiene su sede en Bruselas y todos sus miembros cubren el 80% de la demanda de equipamiento eléctrico en Europa. Las funciones de la asociación son básicamente el soporte para la preparación de normas unificadas y la definición de test y requisitos de homologación que garanticen la calidad y la compatibilidad de los productos. Las tareas principales más detalladas de esta asociación son:

- Fijar las directrices técnicas para el sistema y los productos EIB, así como establecer los procedimientos de ensayo y certificación de calidad.
- Distribuye el conocimiento y las experiencias de las empresas que trabajan sobre el EIB. Encarga a los laboratorios de ensayo las pertenecientes pruebas de calidad.
- Concede a los productos EIB y a los fabricantes de estos una licencia de marca EIB con la que se podrán distribuir los productos.
- Colabora activamente con otros organismos europeos o internacionales en todas las fases de la normalización y adapta el sistema EIB a las normas vigentes.
- Lidera el proceso de convergencia de los tres buses europeos de más amplia difusión como son el propio EIB, el Batibus y el EHS (European Home System).

Se trata, además, de un sistema abierto bajo las mismas premisas que otros sistemas de comunicación como pueden ser el *FieldBus o Bus de Campo*

abierto que son tecnologías de comunicación y protocolos utilizados para la automatización y control de procesos industriales **[6]**. Tanto las especificaciones del protocolo como los procedimientos de verificación y certificación están disponibles. Existen unos componentes críticos del sistema como son los microprocesadores específicos con la pila del protocolo y electrónica de acoplamientos del bus, de los cuales también se puede disponer.

Existen tres posibles medios físicos para la interconexión de los dispositivos EIB: cable de par trenzado, red eléctrica de baja tensión y en un futuro está previsto el desarrollo de los dispositivos por radio-frecuencia. La diferencia entre los dispositivos de cada tipo radica en la electrónica de acceso al medio, siendo el resto del protocolo de comunicaciones común a todos los tipos.

Al igual que otros sistemas domóticos existentes en el mercado, EIB permite la integración de funciones básicas requeridas en viviendas y edificios.

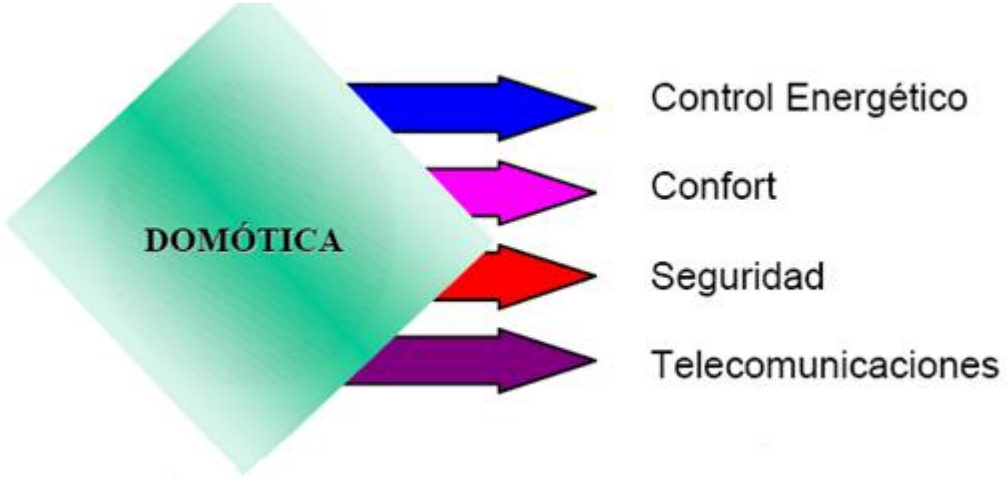


Figura 3 Funciones básicas en las instalación domóticas

Vamos a ver cada una de estas funciones **[5]**:

- **Gestión de la energía**. La finalidad es satisfacer las necesidades del hogar al mínimo coste. Optimización del consumo eléctrico y climatización como pueden ser modos de tarificación nocturna, prevención de situación de consumo innecesario como corte de la calefacción con las ventanas abiertas y otros.
- **Seguridad.** Actualmente es la función más desarrollada y puede integrar simultáneamente múltiples aplicaciones, especialmente si se encuentra integrada dentro de un sistema domótico. Se distinguen dos tipos de seguridad: bienes y persona.

Dentro de seguridad de bienes las principales aplicaciones son aviso a distancia, detección de intrusos o alarmas técnicas. En la seguridad de personas, se han incluido aplicaciones como alumbrado automático, desactivación de enchufes, manipulación a distancia de interruptores, avisos telefónicos, fugas, etc.

- **Confort.** El uso de un sistema integrado de comunicaciones permite disponer de comodidades diversas como el control por mando a distancia, programación de escenas y automatización de tareas como subida y bajada de persianas. Principalmente va dirigido a las instalaciones CVC (climatización, ventanas y calefacción), aunque también presentan confort otro tipo de comunicaciones como los que hemos comentado anteriormente.
- **Comunicación.** Es posible la conexión con el sistema a distancia sin necesidad de estar en el edificio o en la casa, de forma que se pueda modificar y conocer el estado de funcionamiento de la instalación. En este campo está produciéndose un verdadero avance en los últimos años ya que muchos fabricantes están comercializando componentes que permitan el acceso o control a través de las últimas tecnologías existentes en el mercado como pueden ser el control por Internet o mediante teléfonos móviles a través de SMS o WAP.

Además, EIB también presenta las ventajas inherentes a este tipo de sistemas frente a las instalaciones convencionales o tradicionales:

- Reducción del cableado y los costes asociados a la instalación.
- Integración de diferentes funciones en un solo sistema como puede ser control de ventanas, funciones de alarmas, control de persianas y toldos, control de iluminación, etc. Mientras que en los sistemas tradicionales tenemos que disponer de un sistema para cada función que queramos implementar.
- Flexibilidad para ampliaciones y modificaciones futuras. Es posible reprogramar el funcionamiento de la instalación conectando un ordenador al sistema o incluso a distancia mediante un enlace telefónico o a través de internet. Sin embargo, en los sistemas tradicionales tendríamos que modificar la instalación o volverla a hacer de nuevo.

2.3 TECNOLOGÍA

EIB es un sistema descentralizado ya que no requiere de un controlador central de la instalación, en el que todos los dispositivos que se conectan al bus de comunicaciones tienen su propio procesador y electrónica de acceso al medio.

En una red EIB podemos encontrar cuatro tipos de componentes: módulos de alimentación de la red, acopladores de línea que interconectan los diferentes segmentos de la red y los elementos que pueden ser sensores o actuadores.

Los sensores son los encargados de detectar cambios de actividad en el sistema como por ejemplo movimientos, cambio de luminosidad, temperatura, humedad, etc. Cuando se producen estos cambios los sensores transmiten unos mensajes llamados telegramas a los actuadores, los cuales se encargaran de ejecutar los comandos adecuados. Visto de otra manera los sensores funcionarán como entradas al sistema mientras que los actuadores como salidas para la activación y regulación de cargas.

Las instalaciones de tipo EIB pueden direccionar más de 14000 de estos dispositivos, por lo que son aplicables a edificaciones desde viviendas unifamiliares hasta grandes edificios como hospitales, hoteles, oficinas, etc.

2.4 ARQUITECTURA EIB

Está basado en el modelo de referencia *Open System Interconnection (OSI)*, cuyo estándar se publicó en 1984 para generalizar un modelo para el funcionamiento de las redes de telecomunicaciones. Este modelo especifica las funciones requeridas para las comunicaciones de datos y la interacción entre estas funciones.

Figura 4 Modelo OSI

El modelo OSI divide las tareas complejas de las comunicaciones de datos en 7 subtareas definidas, llamadas capas **[7].** Cada capa interactúa con la capa de encima y la de abajo. Una capa, el proveedor de servicios, provee un servicio a la capa inmediatamente más arriba, el usuario de servicio. La interfaz entre ambas capas define como el usuario del servicio puede acceder al servicio del proveedor de servicios, especifica los parámetros y el resultado que espera. Un protocolo define un conjunto de reglas y convenciones que están siendo usadas por capas del mismo nivel, permitiendo la comunicación entre los distintos mecanismo.

El protocolo define el procedimiento, además de la semántica y sintasis de las unidades de datos llamadas *Protocol Data Units (PDUs)*. Estas unidades están formadas por *Protocol Control Information (PCI)* que es un encabezado, y de los datos que van a ser transmitidos que forman lo llamado *Data Unit (DU).*

La implementación de EIB se ha basado en el modelo de OSI implementándose por capas y funcionalidades, con la excepción de que en este protocolo se puede prescindir de alguna capa o nivel por no ser necesaria ya que este protocolo no dispone o no necesita efectuar dichas funcionalidades **[9]**.

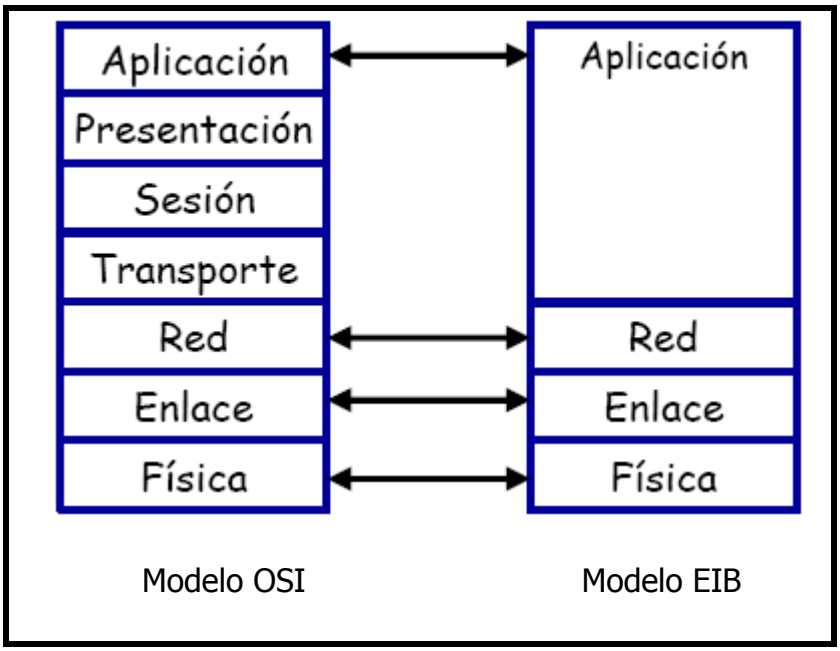


Figura 5 Comparación modelo OSI y modelo EIB

Centrándonos en la comunicación entre puntos, hay 4 servicios primitivos diferentes: solicitud (req), respuesta (res), indicación (ind) y confirmación (con). Los servicios no necesitan siempre hacer uso de tales servicios primitivos y pueden ser clasificados de la manera siguiente:

Servicios localmente confirmados forman parte de una solicitud, una indicación y una confirmación. El usuario de servicios local llama al proveedor de servicios de la capa N. La solicitud y la correspondiente trama de protocolo EIB se generan y se transmiten a la capa (N-1) hasta que es entregada al medio físico. En el lado receptor, la capa N se activa con una indicación, la trama adjunta se decodifica y los datos pasan a la capa anterior.

Capa N en el lado emisor recibe una confirmación de la capa local (N-1) indicando si la capa subyacente es capaz de procesar la solicitud como corresponde.

Servicios confirmados también se componen de una solicitud, una indicación y una confirmación. Con un servicio confirmado, la capa remota genera un asentimiento inmediatamente después de recibir una indicación. En el lado del que envía el asentimiento recibido es pasado hacia la capa N como una confirmación.

Servicios de respuesta siempre consiste en una solicitud, una indicación, una respuesta y una confirmación. La respuesta de confirmación se está generando en el proveedor de nivel de servicio a distancia como una respuesta que se recibe en la capa de carácter local como una confirmación.

2.4.1 Capa de Aplicación. Nivel 7.

La capa de aplicación del modelo de OSI provee funcionalidades que pueden ser usadas directamente por los programas de aplicación. Esta capa ayuda a que los programas de aplicación o aplicaciones entiendan, comprendan o sepan interpretar los datos que se han intercambiado.

En el caso del EIB es una comunicación orientada a bus cuya finalidad es el cumplimiento de funcionalidades específicas de cada dispositivo. El propósito de las comunicaciones puede ser por ejemplo el estado de los interruptores de un grupo de objetos, intensidad luminosa o la temperatura en una habitación.

El programa de aplicación de un sensor mide la cantidad física y escribe el valor de esa cantidad en un objeto de comunicación apropiado. Al mismo tiempo este envía el nuevo valor del propósito de la comunicación a través del bus EIB para informar a los mecanismos pertinentes. El programa de aplicación del actuador es informado a través de la capa de aplicación del actuador. El programa lee el valor que le ha sido enviado por el objeto y ejecuta la correspondiente función de acuerdo a lo que le han enviado. Un ejemplo de este tipo de comunicación puede ser la reducción de la intensidad de una lámpara, apagar o encender las luces según el nivel de luminosidad externa, etc.

2.4.2 Capa de Presentación. Nivel 6.

Esta capa del modelo de OSI tiene la tarea de proteger o aislar la capa de aplicación y el programa de aplicación de tener que tratar con las diferentes formas de representación de la transmisión de datos. Lo que hace es interpretar y, si fuera necesario, ajustar o convertir la sintaxis del mensaje. Adecua los mensajes en el caso de comunicaciones entre objetos o equipos que utilicen una representación interna distinta.

En EIB el problema de convertir la forma de representación en la transmisión de datos no existe, por lo tanto EIB no utiliza capa de presentación.

2.4.3 Capa de Sesión. Nivel 5.

La principal tarea del nivel 5 según el modelo de OSI es el control de las comunicaciones entre dos procesos o dos partes de comunicación. Establece, gestiona y finaliza las conexiones entre dos procesos o aplicaciones haciendo un control de la sesión y verificándola para que en caso de interrupción se pueda reanudar esta conexión.

En el caso de EIB, como los mensajes que se intercambian son cortos los buses no necesitan comunicación de control. Por lo tanto no hay capa de sesión.

2.4.4 Capa de Transporte. Nivel 4.

Según OSI, la capa 4 generalmente tiene la tarea de proteger la capa 5 de las diferentes características de las redes que forman la red general. Es la capa encargada de efectuar el transporte de los datos desde el origen al destino, independizándolo del tipo de red física que se esté utilizando. También se asegura de que el mensaje llegue correctamente al otro lado sin que se pierda en el camino.

Puede tener dos tipos de implementación: orientada a conexión y no orientada a conexión. La implementación orientada a conexión garantiza que el mensaje llega correctamente a su destino haciendo uso de ACK y NACK para saber si la trama ha llegado a su destino o no. La implementación no orientada a conexión hará el mejor esfuerzo para que el mensaje llegue a su destino, sin embargo, no garantiza que la comunicación se realice sin errores y que todo llegue correctamente.

En el caso de la red EIB, el modo normal de operación es la comunicación no orientada a conexión ya que se utilizan direcciones de grupo. Por lo tanto EIB no dispone de nivel 4 ya que no hace falta que se asegure de que los datos han llegado bien al otro extremo.

2.4.5 Capa de Red. Nivel 3.

La principal función de la capa de red es encontrar caminos convenientes o adecuados para la transmisión de los datos, enrutando por los enlaces y asegurando que los paquetes de datos van directos hacia el destino.

En el caso de EIB tenemos el bus dividido en áreas y líneas, indicándose a través de las direcciones físicas. Se utilizan acopladores para reducir la carga del bus, siendo estos capaces de filtrar los telegramas mirando la dirección de destino del telegrama. Permiten el paso de los telegramas a través de esa área o línea siempre y cuando vaya dirigido a esta. En caso contrario, directamente lo desecha. Esto es lo que llamamos enrutamiento en EIB.

Como esta capa tiene una funcionalidad especifica en EIB, se utiliza el nivel 3 de red.

2.4.6 Capa de Enlace de Datos. Nivel 2.

La capa de enlace de datos se ocupa del direccionamiento físico, de la topología de la red, acceso a la red, notificaciones de errores, distribución ordenada de las tramas y del control de flujo.

Esta capa asegura la transmisión entre dos mecanismos. Hay dos funciones básicas de esta capa: primero la composición de la estructura de datos basada en la información de la capa de arriba y chequeo de error (LLC, Control lógico de link) y segundo la codificación de la estructura, asegurando que los datos son transmitidos correctamente (MAC, Control de acceso al medio)

En EIB los mensajes o PDU's de esta capa se llaman *Layer Protocol Data Unit (L_PDU)* y además de contener la información a enviar generalmente proporcionan información como sincronización, secuencia de números, campo de chequeo para la corrección de errores. El formato del mensaje es:

Control field	Source address (physical address of the bus coupling unit)	Receiver address	N_PDU	Check field
8 bit	16 bit	17 bit		8 bit

Figura 6 Formato mensajes EIB

El campo de control asegura que en algunos puntos por donde debe pasar la trama en el bus de línea no sean malinterpretados como bits de empiece. En EIB el D0 se envía el primero.

Con respecto al control de flujo, telegramas que contengan mayor número de ceros serán prioritarios con respecto a otro que tenga menos. En el caso de que suceda una colisión, los telegramas de mayor prioridad son los que se retransmiten primero. El uso de un bus acoplado magnéticamente es el que permite realizar el control de flujo, en la que los ceros anulan a los unos, dando prioridad más alta a la trama que mayor número de ceros contenga.

Utiliza CSMA/CA, explicado detalladamente en la *sección 2.8*, para el control de acceso al bus. La información del estado del bus es llevada desde la capa 2 a la capa 1 siendo los dos estados del bus: bus disponible o colisión.

2.4.7 Capa Física. Nivel 1.

Esta es la capa más baja del nivel de OSI y tiene que ver con la naturaleza de las señales. Tiene la tarea de identificar los bits recibidos de la capa 2 y convertirlos en señales físicas como voltajes, corrientes y ondas electromagnéticas y finalmente transmitir estas señales a través del medio de transmisión.

Esta capa contiene o tiene definida el cable y la instalación eléctrica entre mecanismos y contiene las especificaciones de los componentes electromecánicos. En el caso de EIB hay una gran distinción entre los diferentes medios físicos que se pueden utilizar teniendo distintas características técnicas.

KNX/EIB incluye varios medios de comunicación pero este caso nos centraremos en el que vamos a utilizar que es el par trenzado (KNX TP-1).

La topología puede escogerse libremente exceptuando los lazos que no están permitidos. Los empalmes en el cableado están permitidos y se usan frecuentemente en las implementaciones reales.

2.5 ESPECIFICACIONES TÉCNICAS

2.5.1 Características técnicas del Bus

Las principales características técnicas del Bus a tener en cuenta son las siguientes:

Tipo	Par trenzado, dos pares
Resistencia de la línea	Máx. 37 Ω/km (lazo 74 Ω/km)
Carga de la línea	Máx. 100 nF/km (800 Hz)
Número de vueltas	min. 5/m
Diámetro de la línea	0.8 mm

Tabla 1 Características técnicas bus EIB

2.5.2. Datos técnicos del segmento eléctrico

- Topología aleatoria.
- Capacidad total de un segmento:
  - ✓ sin unidad de acoplador al bus, acoplador de línea ni repetidor de línea: 100 nF máx.
  - ✓ Con unidad de acoplador al bus, acoplador de línea y repetidor de línea: 120 nF máx. (a 10 KHz).
- Resistencia del bus de línea entre el suministro de energía y la unidad de acoplador al bus, acoplador de línea o repetidor de línea: 25Ω máx.

  Resistencia del bus de línea entre dos unidades de acoplador al bus, acoplador de línea o repetidor de línea: 50Ω máx.
- Mínima resistencia entre dos suministradores de energía: 15Ω.
- Mínima longitud del bus de línea entre dos suministradores de energía: 200m.
- Voltaje de caída en un bus de línea entre el suministrador de energía y el acoplador al bus o acoplador de línea: 5V máx.

  Estos criterios determinan como los mecanismos de bus pueden ser físicamente organizados en la línea de bus o segmento. Esto no permite, por ejemplo, tener una aglomeración de 64 dispositivos instalados en un extremo del bus y en el otro extremo el suministrador de energía.
- Máxima longitud de un segmento en el bus de línea: 1000 m.
- Máxima longitud entre dos dispositivos: 700m.
- Máxima longitud entre el suministrador de energía y un dispositivo. 350m.

- La unidad de acoplador al bus (BCU) son suministradas con un valor de voltaje de 24V DC a través del bus.
- Máximo número de dispositivos en un segmento eléctrico: 64.
- Velocidad de transmisión: 9.6 Kbps
- Retardo de transmisión: Para asegurar detección de colisión y corrección, el retardo de transmisión no debe exceder de 10μs. Esta condición se cumple automáticamente si se utilizan solo los cables de bus autorizados, se tiene en cuenta la distancia de 700 metros entre dos dispositivos y no se instalan más de 64 dispositivos en un segmento eléctrico.

2.5.3 Método de transmisión

Utiliza multiplexación por división en el tiempo, banda base, simétrico.

La multiplexación por división en el tiempo hace que sea una transmisión asíncrona de los datos. El modo de transmisión es simétrico transmitiéndose los datos sobre el par de conductores que hacen de medio de transmisión. Además, se emplea una transmisión diferencial, que junto con la simetría, garantiza que el ruido afectara de la misma forma a los dos conductores.

2.6 TOPOLOGÍA

Para la interconexión de dispositivos del bus en cada línea se permite cualquier tipo de topología: árbol, estrella, bus o anillo. Esto facilita la instalación en viviendas y edificios, ya que lo único que no se permite es cerrar anillos entre líneas situadas topológicamente en diferentes subredes.

En la topología de conexión de dispositivos tenemos tres niveles de conexionado que son:

- **La línea.** Es la unidad mínima de instalación. En ella se pueden conectar hasta 64 dispositivos siempre que la capacidad de la fuente de alimentación y de la carga máxima por los dispositivos ya existentes nos lo permitan.
- **Área.** Está formada por varias líneas conectadas a una línea principal por medio de acopladores de línea. Cada línea tendrá que tener su acoplador para poder conectarse a la línea principal. El conjunto de estas líneas conectadas a la línea principal forman el área. Se permiten tener hasta 15 líneas en un área, con un total de hasta 960 dispositivos.

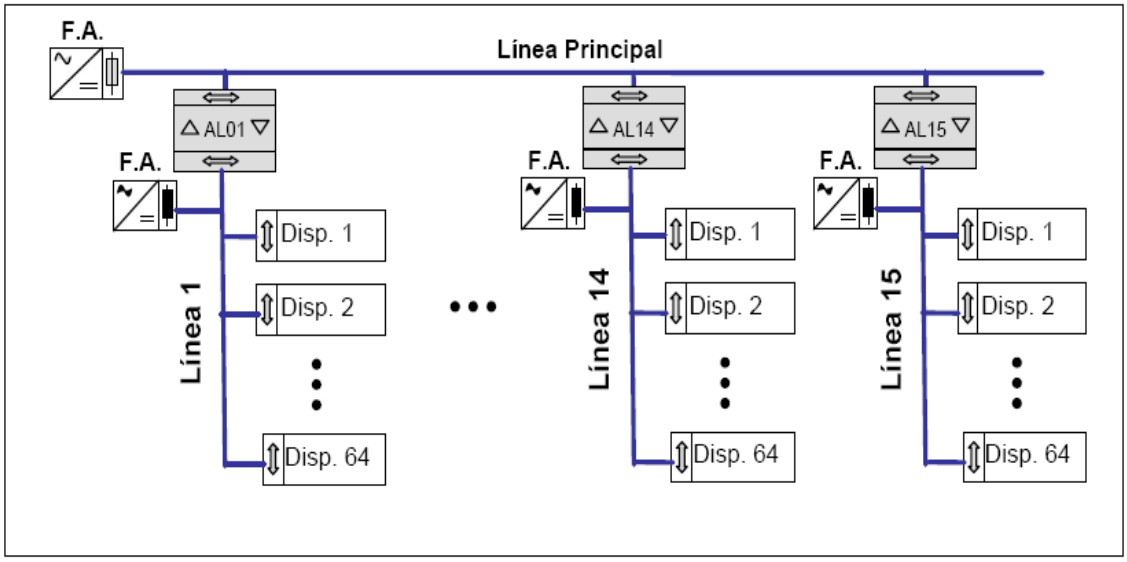


Figura 7 Configuración de un área

Sistema completo. Es el que forman la unión de varias áreas a través de unos dispositivos llamados acopladores de áreas. Se pueden unir un total de 15 áreas lo que nos permitiría integran hasta un máximo de 14.400 dispositivos en un sistema completo.

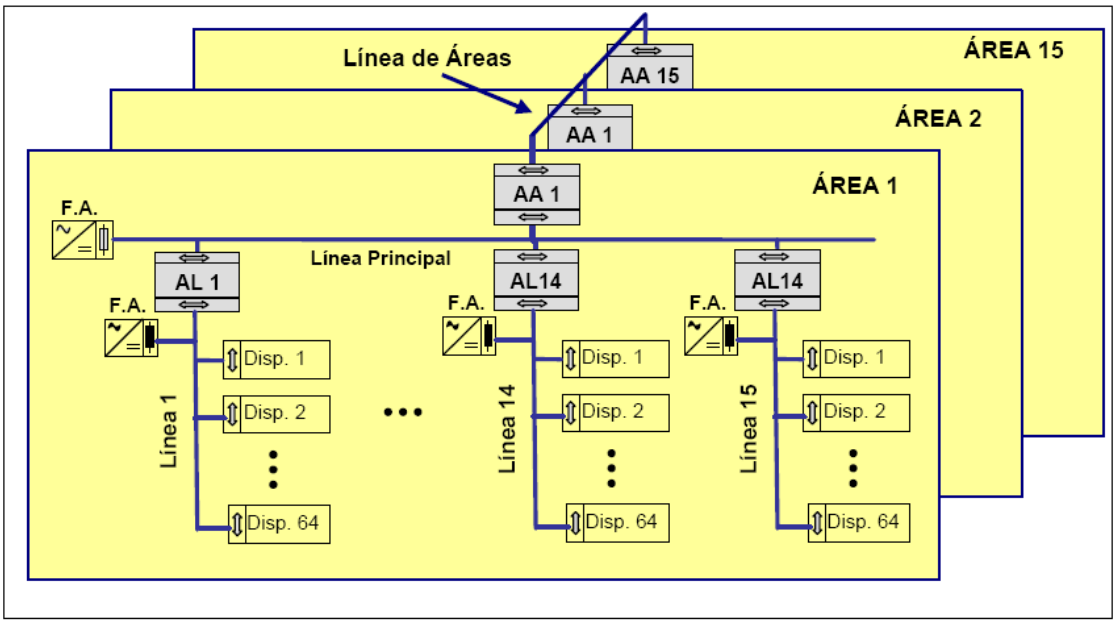


Figura 8 Interconexión de áreas. Sistemas EIB completo

Cada línea, tanto la principal como las secundarias, deben tener su propia fuente de alimentación. Además, la línea principal puede tener conectados directamente hasta 64 dispositivos incluyendo los acopladores de línea.

2.7 DIRECCIONAMIENTO

Los elementos existentes en un sistema EIB quedan perfectamente identificados gracias al sistema de direccionamiento que posee. Existen dos tipos de direcciones: direcciones físicas y direcciones de grupo. Vamos a ver cada una.

2.7.1 Dirección Física

La dirección física identifica unívocamente cada dispositivo y se corresponde con la localización del elemento en la topología global del sistema: área – línea secundaria – dispositivo. La dirección consta de tres campos que están separados por puntos y son:

- **Área (4 bits).** Identifica una de las 15 áreas posibles en el sistema global EIB, siendo el área cero la que corresponde a la dirección de la línea de áreas, es decir, la línea que une todas las áreas o línea principal del sistema.
- **Línea (4 bits).** Identifica una de las 15 líneas posibles en cada área, siendo la línea cero la que se reserva para identificar la línea principal dentro de cada área.
- **Dispositivo (8 bits).** Identifica cada uno de los posibles dispositivos dentro de una línea, siendo el número de dispositivo cero el que se reserva para el acoplador de la línea.

En la siguiente figura se muestra un ejemplo de direcciones físicas de un sistema EIB completo:

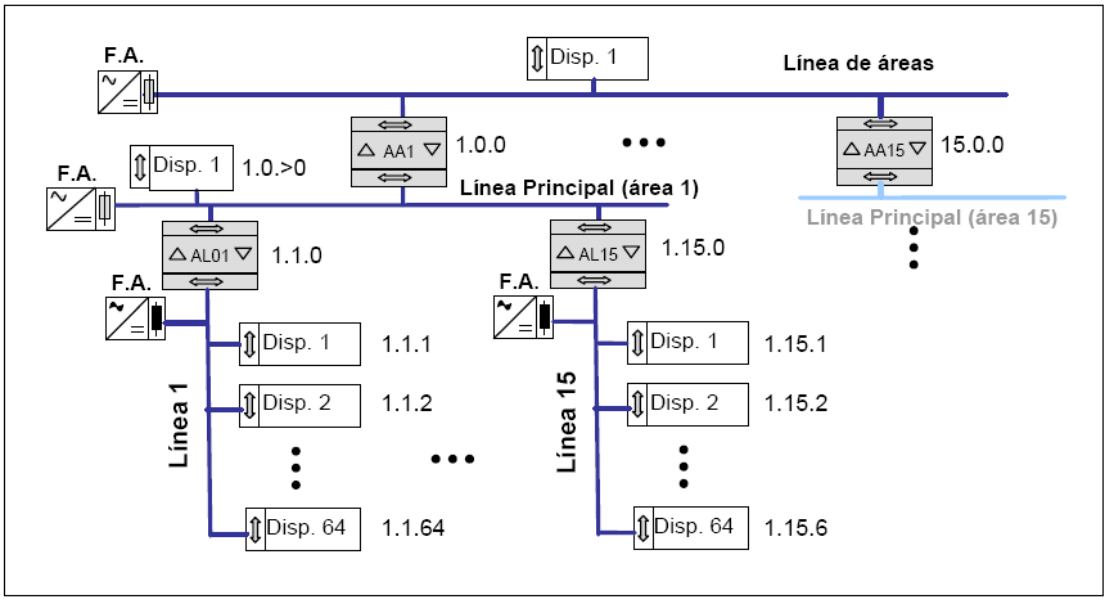


Figura 9 Ejemplo de direccionamiento físico

Para la interconexión de diferentes líneas y áreas se emplea la unidad de acoplamiento. Este es el mismo para los distintos tipos de conexión, y dependiendo de la dirección física que se le asigne actuará como acoplador de línea, acoplador de área, o incluso repetido de una misma línea.

Este acoplador cuando funciona como acoplador de línea o de área, actúa como router o encaminador, y mantiene una tabla interna de direcciones de las subredes que interconecta para aislar el tráfico de ellas. En el caso de que le lleguen mensajes que no estén dirigidos a su subred, desecha el mensaje y no lo envía ya que no va dirigido a ningún elemento perteneciente a su subred.

2.7.2 Dirección de Grupo

Las direcciones de grupo se usan para definir funciones específicas del sistema, y son las que determinan las asociaciones de dispositivos en funcionamiento y la comunicación entre sus objetos de aplicación.

Las direcciones de grupo asignan la correspondencia entre elementos de entrada al sistema que son los sensores y los elementos de salida que son los actuadores.

Se pueden utilizar dos tipos de direccionamiento de grupo: de dos niveles y de tres niveles. El uso de cada tipo se hará dependiendo de las necesidades en la jerarquización de las funciones del sistema.

Las características de cada tipo de direccionamiento son:

- **Dos niveles:** Tiene un grupo principal de 4 bits y un subgrupo formado por 11 bits.
- **Tres niveles:** Tiene un grupo principal compuesto de 4 bits, un grupo medio formado por 3 bits y el subgrupo que consta de 8 bits.

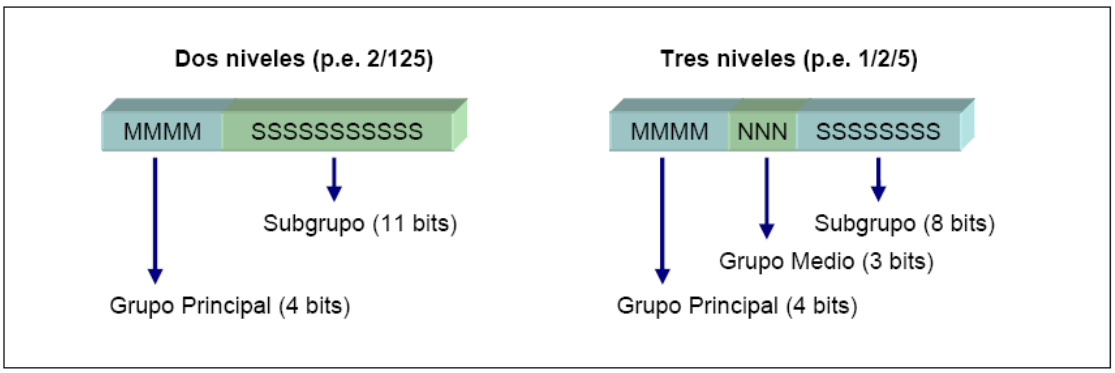


Figura 10 Niveles en las direcciones de grupo

Generalmente el grupo principal se usa para englobar grupos de funciones como alarmas, iluminación, control de persianas, etc. Se pueden utilizar valores de 1 a 13, ya que los valores 14 y 15 no deben emplearse porque no son filtrados por los acopladores y podría afectar al funcionamiento de todo el sistema y al objetivo de los acopladores de aislar su subred. El valor cero está reservado en todos los campos para funciones globales del sistema.

En la configuración de una instalación EIB, la asignación de direcciones de grupo es básica para poder asegurar su correcto funcionamiento. Las direcciones de grupo que son las que relacionan los sensores con los actuadores, se pueden asignar a cualquier dispositivo en cualquier línea ya que son independientes de las direcciones físicas que tenga cada elemento. Hay una serie de condiciones para la asignación de estas direcciones de grupo:

- Los sensores solo pueden enviar a una dirección de grupo, es decir, solo se les puede asociar una solución de grupo.
- Varios actuadores pueden tener la misma dirección de grupo, es decir, responden antes un mismo mensaje o telegrama.
- Los actuadores pueden responder a más de una dirección de grupo, es decir, pueden estar direccionados o asociados a varios sensores simultáneamente.

2.8 MÉTODO DE ACCESO AL MEDIO

El método de acceso al medio que emplea EIB es de tipo *Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA*), cuyas siglas significan "acceso múltiple por detección de portadora evitando colisiones. Es un protocolo de control de redes de bajo nivel que permite que múltiples estaciones o elementos utilicen el mismo medio de transmisión.

La codificación se realiza de modo que el estado lógico '0' es dominante (es un impulso simétrico) sobre el '1' que es recesivo (no hay impulso). El mecanismo de resolución de colisiones es el siguiente:

- El dispositivo comprueba el bus, y si esta libre comienza la transmisión.
- Durante el envío cada dispositivo escucha los datos presentes en el bus, comparándolos con lo que ha transmitido.
- Si no se producen colisiones, el envío se completa sin ningún contratiempo.
- Si, por el contrario, se produce una colisión con los datos enviados por otro equipo, el arbitraje se resuelve por prioridad de los bits dominantes sobre los recesivos. Se retransmitirá la trama que contenga mayor número de ceros en su inicio ya que será más prioritaria que la que contenga menor número de ceros.

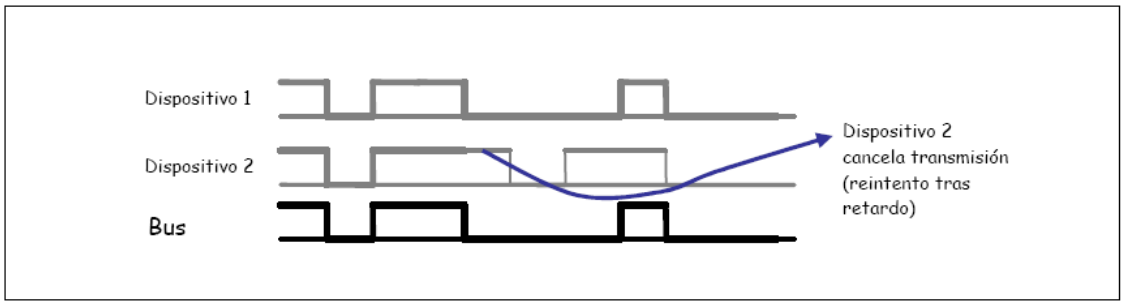


Figura 11 Resolución de colisiones CSMA/CA en EIB

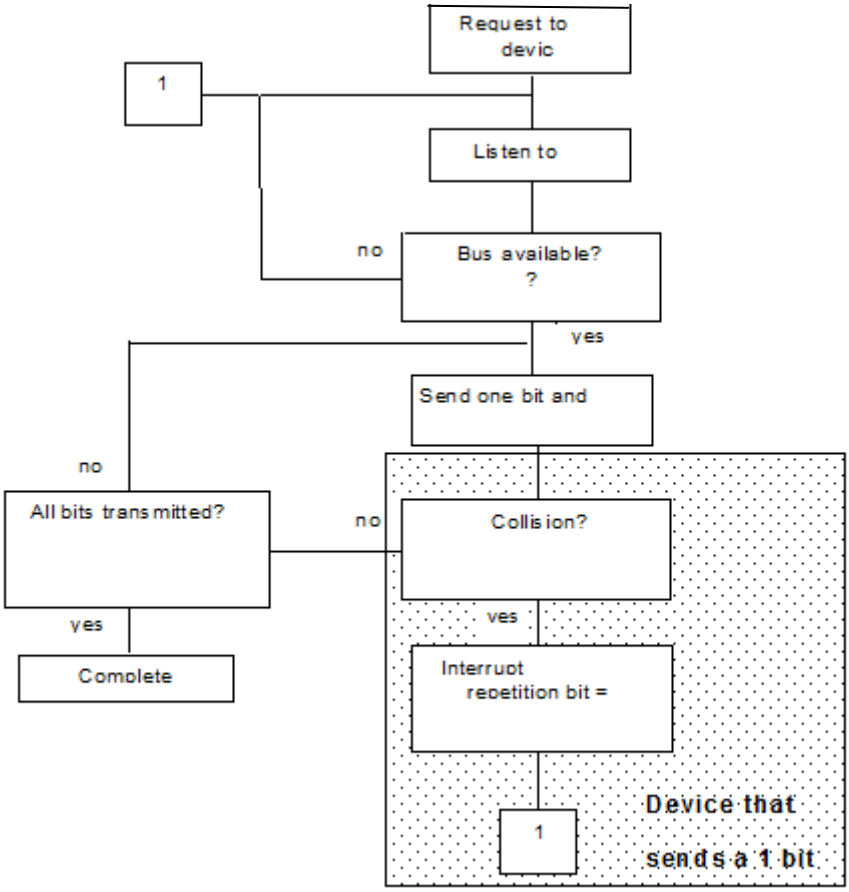


Figura 12 Método de acceso al bus por medio CSMA/CA

2.9 TÉCNICA DE DETECCION DE ERRORES

Utiliza dos tipos de técnicas muy comunes:

- **Paridad (VCR – Vertical redundancy check).** Es un bit por carácter, par o impar; solamente indicado para transmisiones carácter por carácter. Generalmente es aplicado para todos los métodos de

transmisión con parada. Ag menudo nos referimos a este tipo de métodos como paridad vertical. Este método de detección de errores solo detecta errores impares de bits.

- **Paridad horizontal/longitudinal (LRC – Longitudinal redundancy check).** Es un bit por dígito de cada carácter, con lo que tendrá tantos bits como contenga cada carácter. Agrupa todas los caracteres transmitidos y coge el mismo bit de cada carácter y hace la paridad de cada uno, dando como resultando un código de chequeo con el mismo número de bits que tiene la información.

En el caso de EIB utiliza paridad par en la transmisión asíncrona de cada byte, incorporándose dicho bit en el envío del carácter. También crea un byte de comprobación haciendo la paridad impar vertical del mismo byte de cada carácter, y a su vez hace la paridad par de este byte. A continuación se muestra una figura explicativa de cómo realiza EIB la comprobación de errores.

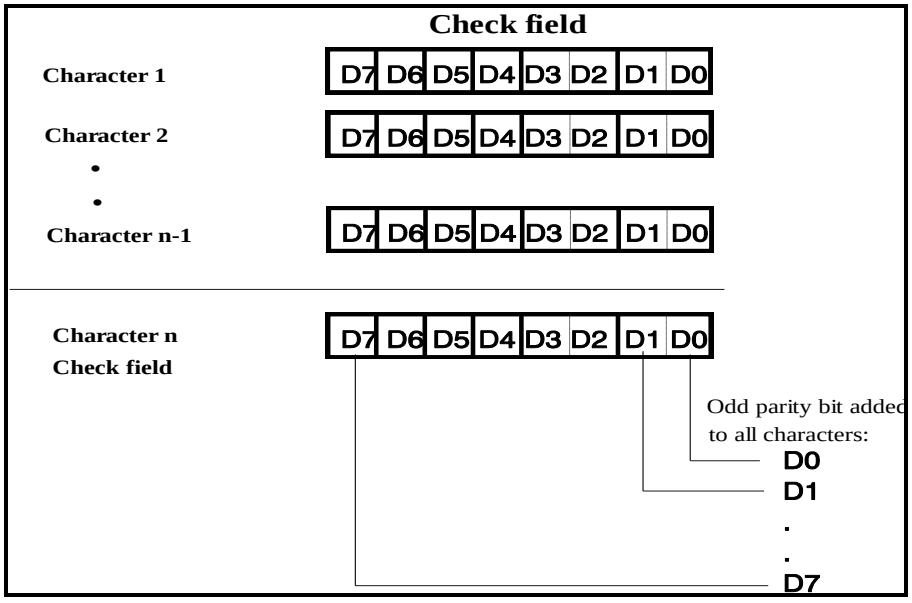


Figura 13 Técnica de comprobación de errores en EIB

2.10 FORMATO DE TELEGRAMA: LPDU

Layer Protocol Data Unit (LPDU) es el mensaje del nivel de Enlace de Datos. Es el utilizado para viajar a través de la red. Además de contener la información a enviar, proporciona información de sincronización, secuencia de número y campo de cheque para la corrección de errores. Dentro de este mensaje van encapsulados los mensajes de los demás niveles.

Para entender un poco mejor, la transmisión de un telegrama en un sistema EIB se realiza cuando se produce un evento como puede ser la

activación de un pulsador o la detección de presencia. El dispositivo emisor, que es el sensor, comprueba la disponibilidad del bus mediante un tiempo t_1 y envía el telegrama. Si no hay colisiones, a la finalización de la transmisión espera un intervalo de tiempo t_2 la recepción del ACK por parte del elemento que ha recibido el telegrama. En el caso de que la recepción del mensaje haya sido incorrecta, se recibe un NACK o bien no se recibe ningún reconocimiento, con lo que la transmisión se reintenta hasta tres veces. Todos los dispositivos que reciben el telegrama, envían el reconocimiento simultáneamente.

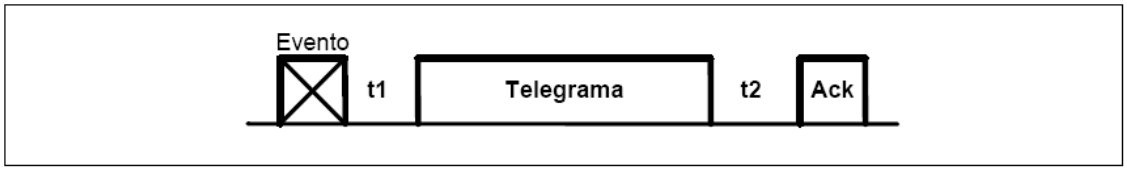


Figura 14 Secuencia de envío de un telegrama ante un evento

Los telegramas se transmiten en modo asíncrono, a una velocidad de 9600 baudios, donde cada carácter o byte consta de 1 bit de inicio, 8 bits de datos, 1 bit de paridad par, 1 bit de parada y una pausa de 2 bits que separa la transmisión de un carácter de la transmisión de otro carácter.

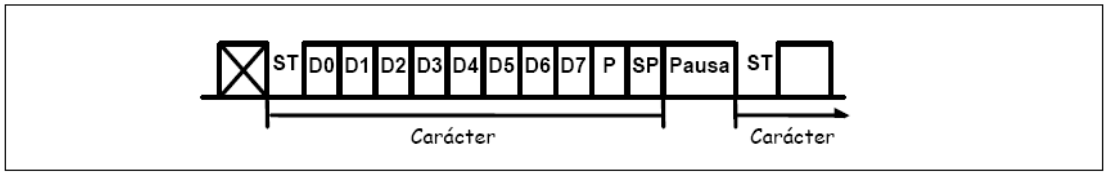


Figura 15 Formato de transmisión de un carácter

De acuerdo con el protocolo, después de la transmisión del telegrama completo habrá un tiempo de espera que por lo menos será de la misma duración de un carácter. El siguiente carácter es reservado para el asentimiento, el cual debe ser enviado por el dispositivo al que le ha llegado la trama. A continuación, tendremos un tiempo ocioso de 50 pulsos de bits que será después de la transmisión del asentimiento.

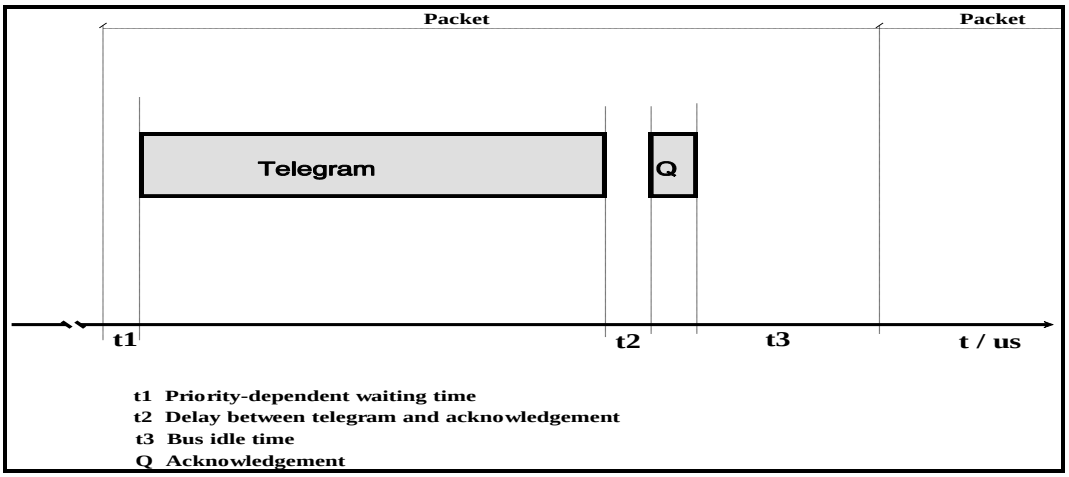


Figura 16 Modo de transmisión del sistema EIB

De este modo la transmisión de un byte supone un tiempo de 1.35 ms, y la de un telegrama completo entre 20 y 40 ms, siendo el tiempo de 20 ms el correspondiente a las ordenes más elementales de marcha-paro.

El telegrama que se transmite por el bus, y que contiene la información específica sobre el evento que se ha producido, tiene la siguiente estructura:

Campo control	Dirección Origen	Dirección destino	N_PDU	Comprobación ó Check
8 bit	16 bit	17 bit		8 bit

Figura 17 Formato LPDU

Campo de control. Tiene la siguiente estructura:

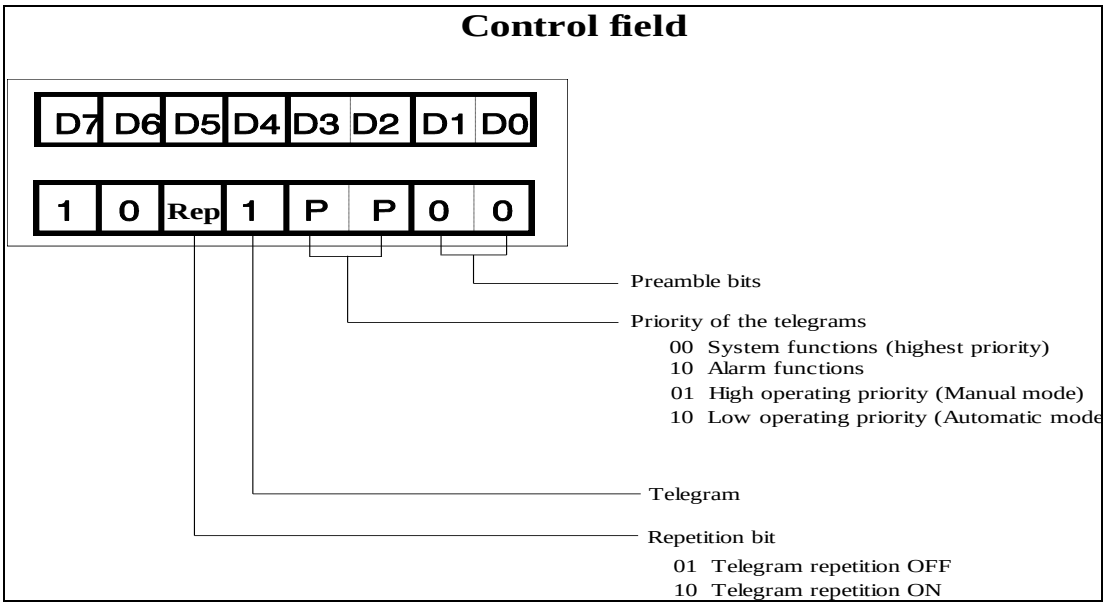


Figura 18 Estructura del campo de Control

Este campo tiene 8 bits incluyendo la prioridad que dicho telegrama tiene para ser enviado según el tipo de función: alarma, servicios del sistema o servicios habituales. El bit de repetición se pone a cero en caso de repetirse algún envío como consecuencia de no recibir el pertinente *Acknowledgment (ACK)* de los destinatarios del mensaje. De este modo se evita que los mecanismos que ya han ejecutado la orden la vuelvan a repetir.

Los cuatros niveles de transmisión son:

1	**0**	**R**	**1**	**P**	**P**	**0**	**0**	**Prioridad de transmisión**
				0	0			Funciones del sistema (prioridad máx.)
				1	0			Funciones de alarma
				0	1			Prioridad de servicio elevada. Modo normal
				1	1			Prioridad de servicio baja. Modo normal
		0						Repetición

Tabla 2 Niveles de transmisión

El término prioridad siempre se refiere a la prioridad definida por la capa 7 de la comunicación de objetos. Esta prioridad es pasada a través de todas las capas hasta la capa 2. Sirve en caso de colisión para determinar qué trama es la más prioritaria. La de mayor número de ceros se transmite como prioritaria.

Dirección de origen. El dispositivo que transmite la trama envía su dirección física con el siguiente formato: 4 bits para el área, 4 bits de identificador de línea y 8 bits de identificador de dispositivo. De este modo se

conoce el emisor del telegrama en las tareas de mantenimiento. Su forma es la siguiente:

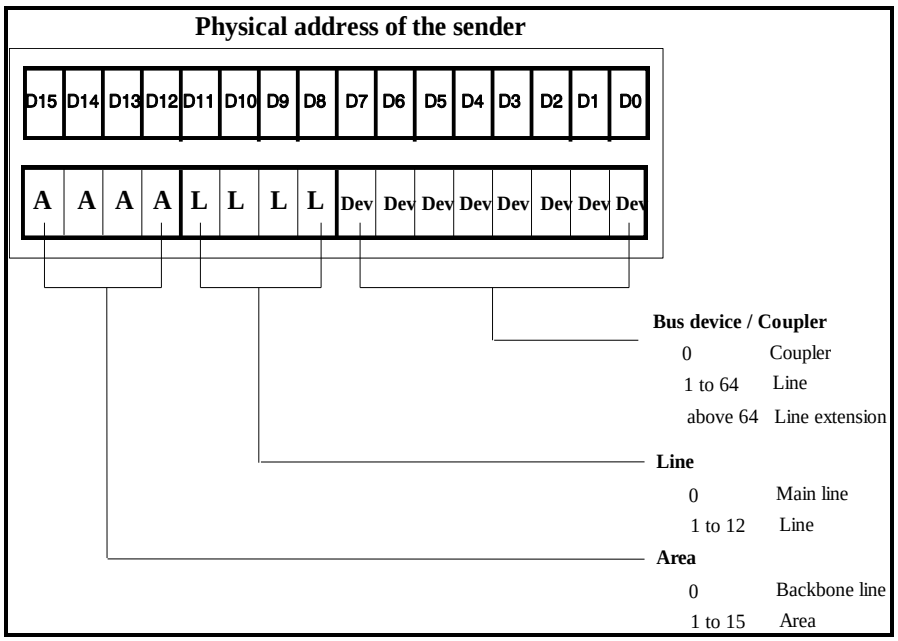


Figura 19 Formato de la dirección origen

Dirección destino. La dirección de destino puede ser de dos tipos, en función del valor que tome el bit de mayor peso de este campo que será el bit 17. Si tiene el valor '0', se trata de una dirección física y el telegrama irá dirigido únicamente a un dispositivo en concreto. Si el valor que contiene este bit es '1' se trata de una dirección de grupo, y el telegrama se dirigirá a todos los mecanismos que tengan esa dirección de grupo. Tiene la siguiente forma:

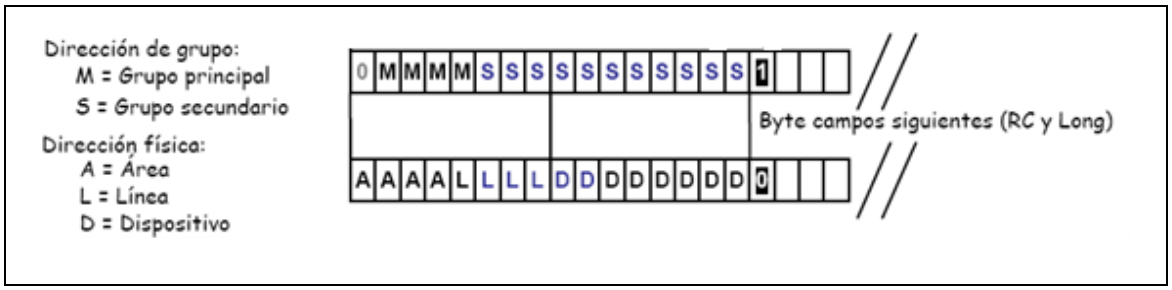


Figura 20 Formato de la dirección destino

Comprobación. Consiste en un byte que se obtiene del cálculo de la paridad longitudinal impar (LRC) de todos los bytes incluidos en el telegrama, obteniéndose cada uno de sus bits a partir del cálculo de la paridad impar de los bits de igual peso en el resto de campos. Este campo es independiente del bit de paridad par que se obtiene al realizar la transmisión en modo asíncrono

de cada byte del telegrama, y se emplea como medida adicional para garantizar una transmisión fiable.

El proceso de cálculo es el que se muestra en la siguiente figura:

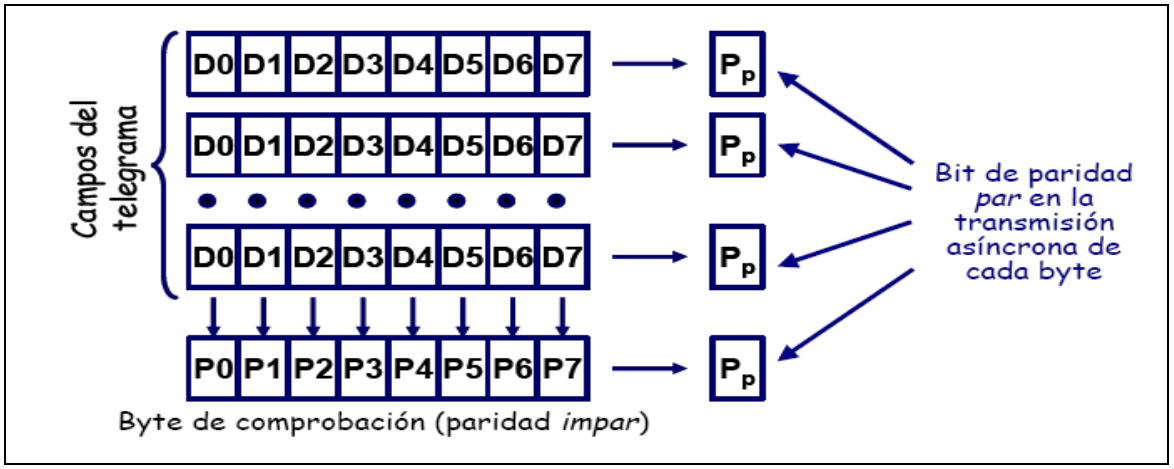
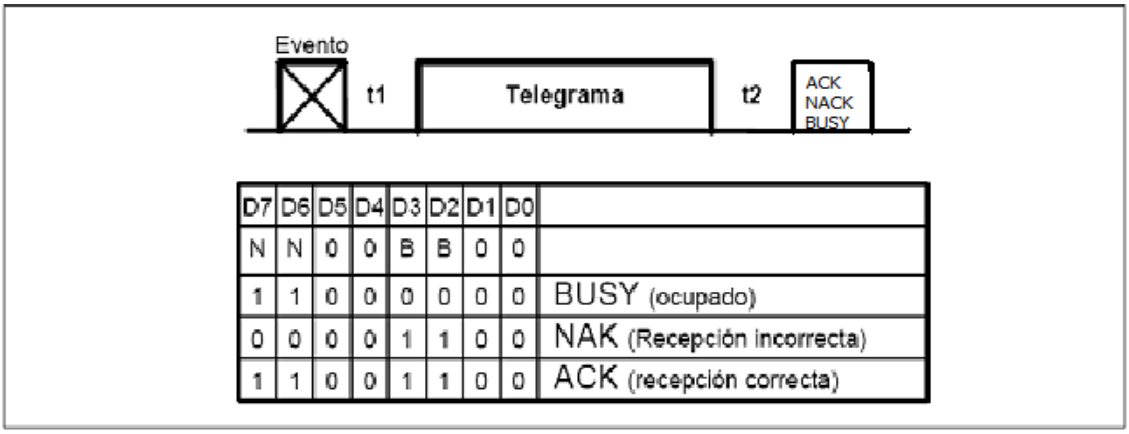


Figura 21 Proceso de cálculo de campo de comprobación

Cuando el dispositivo al que va dirigido el telegrama le llega la trama completa, comprueba si es correcto a partir del byte de comprobación. Hace el mismo proceso que ha hecho el dispositivo emisor, comparando el byte que ha obtenido el con el que ha recibido en el campo de comprobación. Si es correcto, se envía un reconocimiento o ACK, de lo contrario se envía un reconocimiento negativo o NACK para que el emisor repita el envío ya que la trama recibida es incorrecta. Si el dispositivo está ocupado envía un código Busy para indicar al emisor que reintente la transmisión tras un pequeño intervalo de tiempo.

Los tipos de respuestas por parte del elemento que ha recibido el paquete son:



D7	D6	D5	D4	D3	D2	D1	D0	
N	N	0	0	B	B	0	0	
1	1	0	0	0	0	0	0	BUSY (ocupado)
0	0	0	0	1	1	0	0	NAK (Recepción incorrecta)
1	1	0	0	1	1	0	0	ACK (recepción correcta)

Figura 22. Tipos de respuestas de reconocimiento

NPDU. Es la trama correspondiente al nivel de red y tiene la siguiente estructura:

<table>
<tr><td rowspan="3">RC (Contador de enrutado)</td><td rowspan="3">Longitud</td><td colspan="4">T-PDU</td></tr>
<tr><td rowspan="2">T-PCI</td><td colspan="3">A-PDU = T-DATA</td></tr>
<tr><td>A-PCI</td><td>A-DATA/APCI</td><td>A-DATA</td></tr>
</table>

Figura 23 Formato NPDU

RC. Routing counter o contador de enrutamiento. Son 3 bits cuya función es contabilizar por los distintos positivos de enrutado que ha pasado la trama. Solo se utiliza para funciones de enrutamiento.

Longitud. 4 bits que indican cuantos bytes contiene el campo de datos siendo la longitud 0 el correspondiente a un byte y la longitud 15 la correspondiente a 16 bytes.

TPDU. Es la unidad de datos o trama que utiliza la capa de transporte. La capa de transporte lo que hace es identificar el tipo de TPDU correspondiente. Su estructura es la siguiente:

<table>
<tr><td colspan="4">T-PDU</td></tr>
<tr><td rowspan="2">T-PCI</td><td colspan="3">A-PDU = T-DATA</td></tr>
<tr><td>A-PCI</td><td>A-DATA/APCI</td><td>A-DATA</td></tr>
</table>

Figura 24 Formato TPDU

T-PCI. Son 6 bits que indican los distintos tipos de PDU se pueden enviar:

UDT (00)	Paquete de datos	
NDT (01)	Número de paquete de datos	
UCD (10)	Control de paquete de datos	00 OPR (Open) 01 CLR (Close)
NCD (11)	Numero de paquete de datos de control	10 ACK 11 NAK

2 bits 4 bits

Tabla 3 Tipos de T_PCI

Las UCDs pueden ser de dos tipos, abiertas o cerradas, son usadas para el establecimiento o cierre de la conexión y son enviadas siguiendo una petición del servicio BCU.

APDU. Es la PDU de la capa de aplicación, cuyos campos manejan la comunicación de objetos para el programa de aplicación y proceso el grupo de telegramas como funciones de administración. Tiene la siguiente estructura:

A-PCI	A-DATA/APCI	A-DATA

4 bits 6 bits 0 a 14 bytes

Figura 25 Formato A_PDU

Las APDUs más utilizadas con un tráfico normal son las siguientes:

APCI	**Name**
0000	ValueRead. Leer valor.
0001	ValueResponse. Respuesta valor.
0010	ValueWrite. Escribir valor

Tabla 4 Tipos de A_PCI más utilizados

Hay otros tipos de APCI que no son normalmente usadas y son:

APCI	Name
0011	PhysAddr
0100	AddrRequest
0101	AddrResponse
0110	AdcRead
0111	AdcResponse
1000	MemoryRead
1001	MemoryResponse
1010	MemoryWrite
1011	UserMessage
1100	MaskVersionRead
1101	MaskVersionResponse
1110	Restart
1111	Escape

Tabla 5 Tipos de A_PCI

Para el caso de los servicios "UserMessage" y "Escape", los siguientes 6 bits son añadidos para el control de información como una extensión del APCI. Hoy en día existen un total de unos 50 servicios añadidos, que habitualmente no son utilizados.

Datos. Este campo puede contener hasta 15 bytes (desde el byte 0 a byte 14). Estos datos están normalizados de acuerdo con el EIB Interwoking Standart (EIS) **[8]**. En la siguiente tabla se muestran los tipos EIS normalizados junto con su tamaño y función aunque los más utilizados son EIS 1 y EIS 2:

Nº EIS	Función EIB	Nº bytes	Descripción
EIS 1	Conmutación (*switching*)	1 bit	Encendido/apagado, habilitar/deshabilitar, alarma/no alarma, verdadero/falso
EIS 2	Regulación (*dimming*)	4 bit	Se puede utilizar de 3 formas distintas: como interruptor, como valor relativo y como valor absoluto.
EIS 3	Hora (*time*)	3 bytes	Día de la semana, hora, minutos y segundos.
EIS 4	Fecha (*date*)	3 bytes	Día/mes/año (el margen es de 1990 a 2089).
EIS 5	Valor (*value*)	2 bytes	Para enviar valores físicos con representación S,EEEE,MMMMMMMMMMM
EIS 6	Escala (*scaling*)	8 bit	Se utiliza para transmitir valores relativos con una resolución de 8 bit. P.e. FF = 100 %
EIS 7	Control motores (*control drive*)	1 bit	Tiene dos usos: Mover, arriba/abajo o extender/retraer y Paso a Paso.
EIS 8	Prioridad (*priority*)	1 bit	Se utiliza en conjunción con EIS 1 ó EIS 7.
EIS 9	Coma flotante (*float value*)	4 bytes	Codifica un número en coma flotante según el formato definido por el IEEE 754.
EIS 10	Contador 16 bit (*16b-counter*)	2 bytes	Representa los valores de un contador de 16 bit (tanto con signo como sin signo).
EIS 11	Contador 32 bit (*32b-counter*)	4 bytes	Representa los valores de un contador de 32 bit (tanto con signo como sin signo).
EIS 12	Acceso (access)	4 bytes	Se usa para conceder accesos a distintas funciones.
EIS 13	Caracter ASCII (*Character*)	8 bit	Codifica según el formato ASCII
EIS 14	Contador 8 bit (*8b-counter*)	8 bit	Representa los valores de un contador de 8 bit (tanto con signo como sin signo).
EIS 15	Cadena (*Character String*)	14 bytes	Transmite un cadena de caracteres ASCII de hasta 14 bytes.

Tabla 6 Tipos EIS normalizados

El EIS contiene los datos útiles para cada función asignada a los objetos de comunicación. Según este estándar existen siete tipos diferentes, cada uno asignado a un tipo de acción de control. Entre ellos están conmutación, regulación de luz, envío de valor absoluto, envío de valor en punto flotante, etc. De este modo, se garantiza la compatibilidad entre los dispositivos del mismo tipo de los distintos fabricantes existentes en el mercado.

Los objetos de comunicación son instancias de clases definidas en el estándar, y se incluyen en ellos programas almacenados en memoria de los dispositivos para realizar una determinada acción. Normalmente, el programa de aplicación que se ejecuta en un dispositivo dispone de varios objetos de comunicación, que pueden ser de diferentes tipos EIS. Por ejemplo, un pulsador de dos teclas con un programa de control de iluminación puede tener cuatro objetos: dos de conmutación de tipo EIS 1, que envían órdenes de apagado y encendido, y otros dos de regulación de tipo EIS 2 para el envío de órdenes de incremento o decremento de la luminosidad. Las asociaciones de direcciones de grupo se realizan para cada uno de estos objetos de

comunicación. Un componente EIB con una dirección física, contiene varios sensores o varios actuadores, cuyo funcionamiento lógico es independiente.

2.10.1 Ejemplo de telegramas EIB más utilizados

Los tipos EIS más comunes y utilizados hoy en día son los de conmutación y regulación que son EIS1 y EIS2. Vamos a ver estos dos tipos como quedaría la parte de datos a enviar por parte del elemento.

- **EIS1 CONMUTACIÓN.** Utiliza un bit para la información. Las tareas de este tipo de trama son de encendido/apagado, habilitar/deshabilitar, alarma/no alarma, verdadero/falso. La función de conmutación en un actuador es pasar de un estado a otro, aunque también se puede aplicar en operaciones lógicas como funciones de habilitar/deshabilitar algo.

  Los valores que tiene son:

  0 – OFF / deshabilitado / falso / no alarma

  1 – ON / habilitado / verdadero / alarma

  El tipo de trama y el campo de información sería lo siguiente:

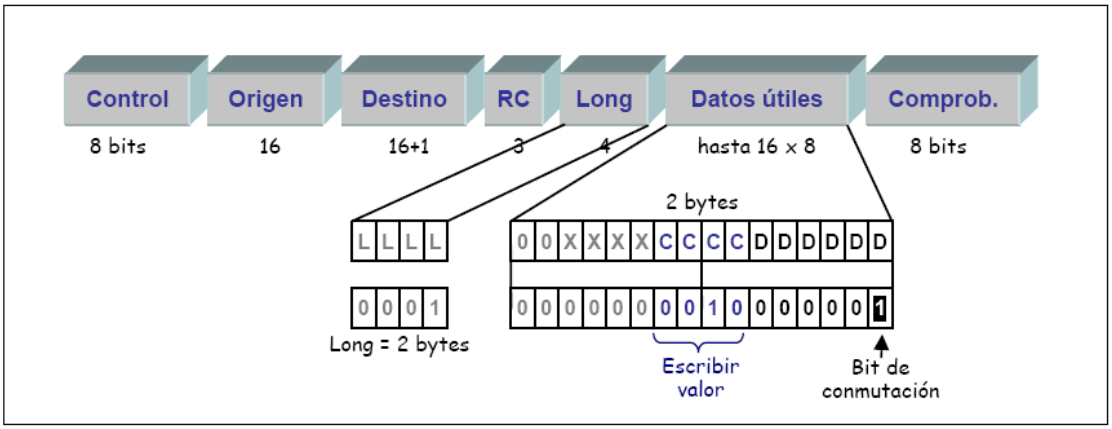


Figura 26 Ejemplo de trama de datos de tipo EIS1

Este tipo de trama es la más sencilla y la que menor número de bit contiene. El envío de este tipo de mensaje tendrá una duración mínima que será de 20ms.

- **EIS2 REGULACIÓN.** Se puede utilizar de tres formas distintas: como interruptor, como valor relativo y como valor absoluto.

El uso de este tipo de mensaje como interruptor tiene la misma finalidad y el mismo contenido que un mensaje de tipo EIS1. Usa un 1 bit para mandar la información.

Como valor absoluto de la regulación de intensidad utilizaría 1 byte completo para establecer directamente el valor de la intensidad de luz en un rango de valores de 0 que sería apagada hasta 255 que sería el máximo de intensidad.

La forma más usada del EIS2 es como valor relativo de la intensidad que utiliza 4 bits para mandar la información. Esta función transmite al actuador de intensidad un comando cuyo contenido es respecto a la intensidad recientemente establecida. Tiene la siguiente estructura:

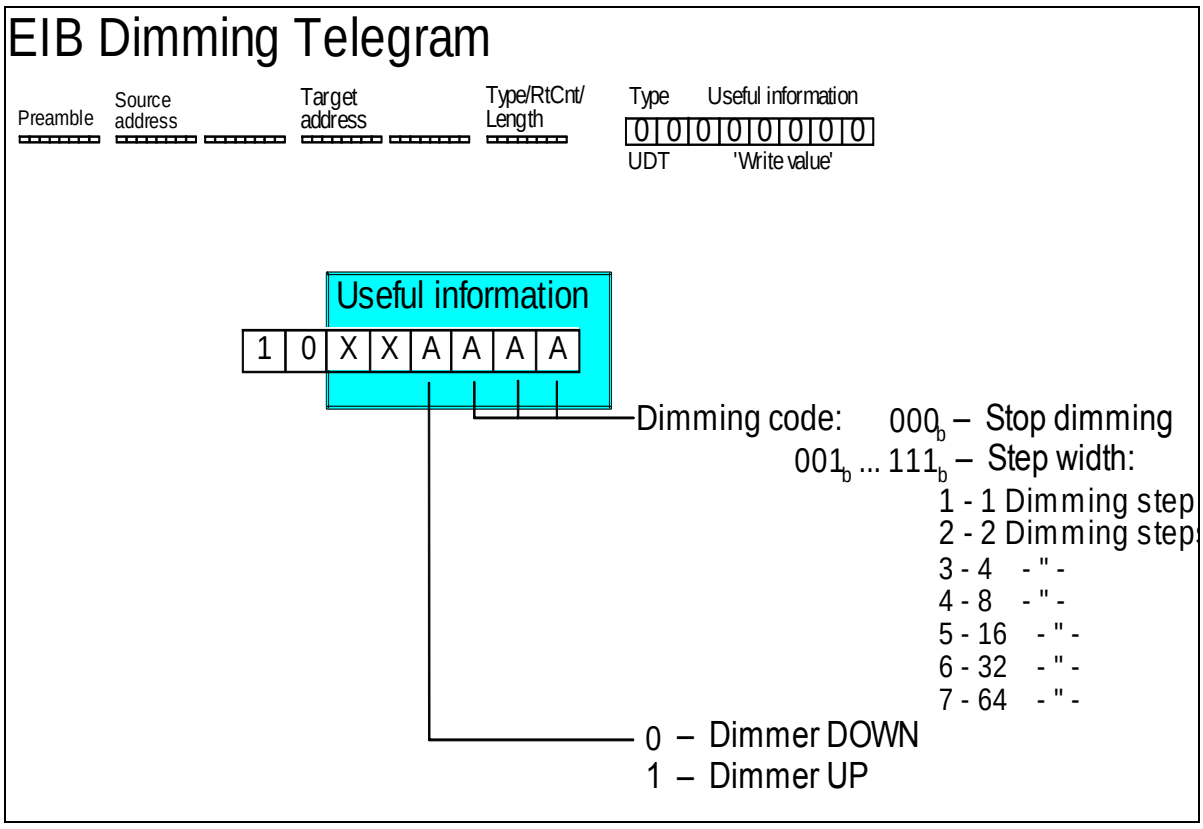


Figura 27 Ejemplo de un telegrama de tipo EIS2

El bit 3 se utiliza para especificar si la nueva intensidad está por debajo o por encima del valor que había anteriormente independientemente del valor de intensidad nuevo. Solamente nos indica si ha bajado o subido la intensidad.

El bit 0-2 especifica el rango de regulación de la intensidad de luz. Para controlar la intensidad hacemos uso de un rango de brillo desde 0% a 100%, el cual variará según unos saltos o escalones de intensidad. El número de escalones es determinado por el código de intensidad que es transmitido como parte del telegrama de regulación de la intensidad.

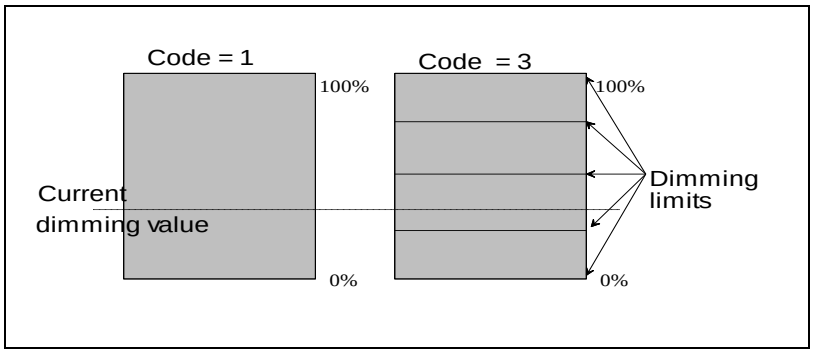


Figura 28 escalones de intensidad

Un actuador de regulación de intensidad puede estar en tres diferentes estados, dependiendo del tipo de telegrama que haya recibido:

Estado	**Descripción**
OFF	El actuador de regulación de intensidad ha sido apagado
ON	El actuador de regulación de intensidad ha sido apagado y está en el valor más bajo posible que se haya establecido
Regulación en proceso	El controlador de intensidad esta encendido, la intensidad es ajustada hacia el punto establecido.

Tabla 7 Estados de un regulador

Un cambio de un estado a otro puede deberse a que se produzca un evento en objeto que realiza el control de la intensidad y manda los mensajes al actuador para que ejecute los comando de encendido, apagado o regulación de la intensidad.

Podemos ver un diagrama de flujo indicándonos los diferentes estados de un actuador de regulación de la intensidad y los diferentes eventos que producen un cambio de estados y que tendremos que tener en cuenta a la hora de programar este tipo de elementos en un sistema EIB.

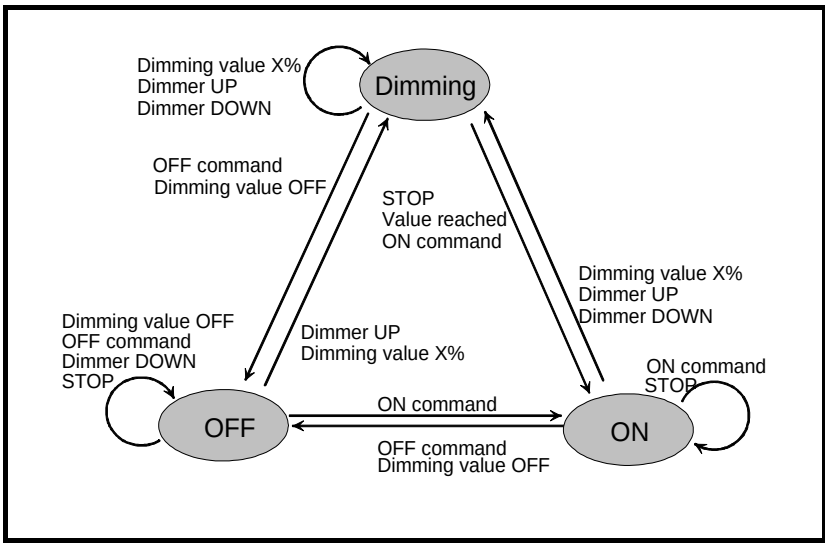


Figura 29 Cambio de estado de un regulador

2.11 COMPONENTES FÍSICOS

Al margen de todos los elementos auxiliares para hacer posible el funcionamiento de un sistema completo EIB, como son la fuente de alimentación, filtros y cables, los elementos más importantes en la instalación son los dispositivos dotados de una cierta 'inteligencia'.

Al tratarse de un sistema distribuido, las funciones están programadas en forma de objetos de aplicación en los sensores y actuadores que intercambian información, posibilitando la realización de las acciones de control. Estos dispositivos constan de tres partes básicas:

- **Acoplador al bus (BCU).** Es donde se encuentra el programa de aplicación. Es un aparato universal, que contiene la electrónica necesaria para gestionar el envío y recepción de telegramas, ejecución de los objetos de aplicación, filtrado de direcciones físicas y de grupo para reconocer telegramas destinados al dispositivo, comprobación de errores, etc. El acoplador examina cíclicamente la interfaz de aplicación para detectar cambios de señal. Consta de un modulo de transmisión y un controlador del enlace al bus.

  Los programas de aplicación están en una base de datos que proporciona cada fabricante, y pueden ser descargados a la BCU a través del bus utilizando el software adecuado.
- **Interfaz de aplicación.** Es un conector estándar de diez pines, de los cuales cinco se utilizan para datos, tres para tensiones de alimentación, y uno para entrada analógica al acoplador al bus que se emplea para la identificación del tipo de dispositivo final en función de una resistencia situada en el mismo.

- **Dispositivo final.**

Existen dos tipos de componentes EIB dependiendo del modo de instalación:

- Componente de carril DIN, con el mismo formato que las protecciones eléctricas como interruptores automáticos o diferenciales.

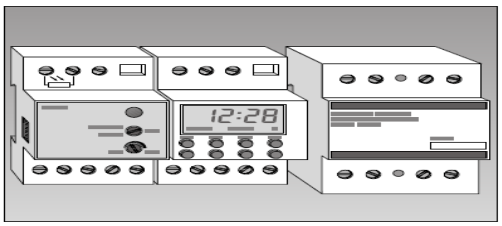


Figura 30 Ejemplo de componente EIB sobre carril DIN

- Componentes a empotrar, para su instalación en cajas universales de empotrar, falso techo o cajas de empalme.

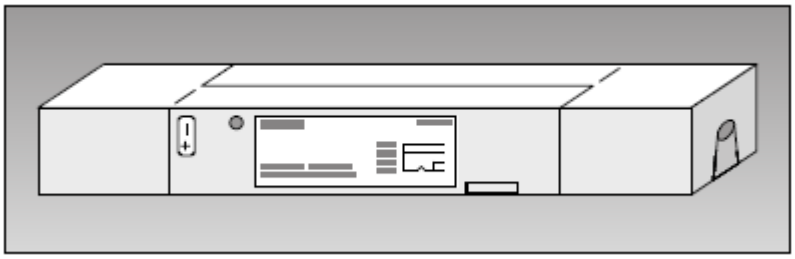


Figura 31 Ejemplo de componente EIB integrado

Los componentes básicos del sistema como la fuente de alimentación, filtro y acopladores sólo están disponibles en la versión de carril, mientras que el resto pueden encontrarse en ambas versiones.

Un componente EIB puede disponer de diversas líneas de entrada-salida, de tipo digital o analógico, con las que realizar diversas funciones como puede ser el ejemplo de un actuador que puede tener cuatro salidas binarias que controla de manera independiente. Cada programa de aplicación tiene definidos una serie de objetos de comunicación que se asocian a cada una de las funciones. Cada objeto se comporta como un dispositivo independiente y tendrá asignadas las direcciones de grupo que lo asocian con otros componentes de la instalación.

2.12 INSTALACIÓN

La realización de una instalación EIB requiere una serie de fases que son **[5]**:

- **Planificación previa** según los requisitos del usuario y del tipo de edificio en el que se vaya a hacer la instalación. El establecimiento de los requisitos ha de hacerse de acuerdo con el cliente de la manera más precisa posible.
- **Diseño del proyecto**. En esta fase se determinan los dispositivos que se van a emplear, material necesario, protecciones y funcionalidad de los componentes. El diseño de la instalación eléctrica se hace en paralelo al del cableado del bus. En esta etapa se tiene que comprobar que todos los elementos que hemos puesto en el diseño y todas las distancias cumplen con las especificaciones del estándar EIB.
- **Instalación eléctrica.** La instalación debe hacer respetando la reglamentación vigente en el área a realizarla.
- **Programación.** Esta es la etapa final de la realización del proyecto. Generalmente se conecta un ordenador personal al bus mediante una pasarela. Se realiza la programación de las direcciones físicas de los dispositivos, carga de los programas de aplicación en los componentes, y programación de las direcciones de grupo. También se realizar la programación de las tablas de filtros en los acopladores de línea y de área. Este proceso se puede realizar en modo local o mediante conexiones a través de la línea telefónica o internet.

2.12.1 Diseño y realización de la instalación

La instalación del cable del bus se suele hacer separada de la de 230V pero en paralelo llegando a todos los elementos existentes en el bus. Los sensores solo se conectan al bus de datos, mientras que los actuadores requieren conexión tanto en el enlace de datos como en la red eléctrica.

Para el cableado del bus se emplea cable apantallado simétrico de dos pares, de los cuales solo se emplea el par rojo-negro, mientras que el otro queda reservado para usos futuros como la transmisión de señales de audio o video u otras aplicaciones de alta calidad.

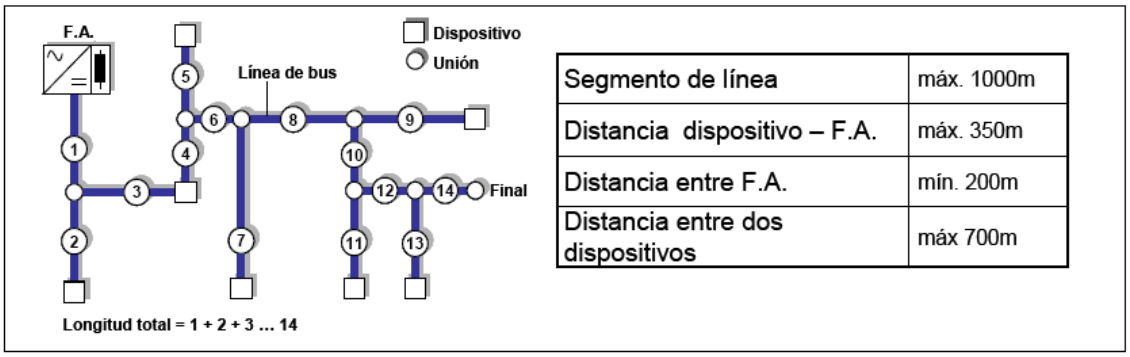
Para la instalación es recomendable utilizar canalizaciones separadas de la red, aunque sería posible ubicarla conjuntamente.

Los paneles de distribución suelen ser comunes, pero separando componentes EIB de protecciones de la red. En estos armarios se alojaran los componentes EIB de carril DIN.

La distribución del cableado se puede realizar de diversas maneras, empleando conectores de empalme de inserción automática tanto para empalmes como para la conexión de los dispositivos.

Una vez realiza la instalación se tendrá que proceder a la comprobación teniendo los siguientes puntos o fases en cuenta:

- Verificar las longitudes permitidas. Se tienen que respetar las longitudes de las especificaciones técnicas de EIB



Segmento de línea	máx. 1000m
Distancia dispositivo – F.A.	máx. 350m
Distancia entre F.A.	mín. 200m
Distancia entre dos dispositivos	máx 700m

Figura 32 Especificaciones sistema EIB

- Comprobar el marcado de los cables para evitar la confusión con otro cableado existente en el lugar de la instalación.
- Comprobar conexiones incorrectas.
- Comprobar la resistencia de aislamiento del cable.
- Comprobar la polaridad
- Medir la tensión en cada extremo del cable.

2.13 EIBSEC: EXTENSIÓN DE SEGURIDAD DE KNX

Debido al aumento de la demanda, la ampliación de KNX hacia nuevas áreas de aplicación como infraestructuras sensibles, edificios de gobierno, etc., surge un requisito: un sistema de control subyacente que sea fiable y robusto ante manipulaciones maliciosas de seguridad **[3].**

Por esta razón, es necesaria la protección de los datos en el proceso de intercambio, así como impedir el uso no autorizado de los servicios de gestión que se utilizan para propósitos de configuración y mantenimiento. EIBSec soporta mecanismos para garantizar la confidencialidad de los datos e integridad, así como un servicio de autenticación. La principal característica y más importante que de EIBSec es que mantiene compatibilidad con el estándar KNX.

Uno de los principales motivos de la protección contra los ataques de seguridad es mejorar las instalaciones ya existentes, ya que un ataque de seguridad en un servicio ya instalado y en funcionamiento puede tener un impacto económico muy elevado y significativo.

Una vez que el protocolo de seguridad ya está instalado, dando una protección contra el acceso no autorizado en la red y la seguridad del sistema no depende del medio físico, se puede incluir otras tecnologías como por ejemplo la inalámbrica sin comprometer la seguridad de la instalación.

Con esto lo que se pretende también es incluir una amplia gama de aplicaciones usando el protocolo KNX y no limitarse exclusivamente a lo ya existente, utilizando sistemas totalmente integrado. Gracias a esto, KNX refuerza aún más su posición en el mercado con respecto a la competencia del sistema en tecnologías.

2.13.1 Componentes específicos de EIBSec.

Los elementos que hacen la funcionalidad específica de EIB son los acopladores, llamados *unidades de acoplamiento avanzadas ACU.* Estos dispositivos se componen de dos bloques diferentes:

- *Unidad de acoplamiento:* Es responsable de enrutar el trafico de red. Se implementa la funcionalidad de un acoplador estándar KNX.
- *Unidad de Gestión de claves:* Proporciona los servicios necesarios de gestión de calves como la generación de claves y la distribución, revocación y limitación de vida de la clave.

Cada ACU es responsable de mantener la clave secreta de su segmento de red. Si un dispositivo quiere recuperar la clave secreta de su segmento de red debe solicitarla a la ACU.

2.13.2 Comunicación Segura

Para la comunicación segura EIBSec utiliza AES para cifrar los datos. Dispone de dos modos de cifrado diferentes:

- *Modo normal:* solo se utiliza durante el establecimiento de la sesión y recuperación de clave de grupo. Solo se lleva a cabo el cifrado de los datos de usuario.
- *Modo contador:* se utiliza para la transmisión de la gestión y el proceso de los datos una vez que está establecido el canal seguro. Para evitar modificaciones no autorizadas del mensaje se hace una suma de comprobación CRC de 32 bits añadiéndose a los datos de usuario. Una vez añadido esto, se hace uso de un contador de 128 bits para hacer XOR con el mensaje. El resultado es cifrado con AES128.

2.13.3. Mensajes EIBSec

Dependiendo de modo de comunicación utilizada, existen dos tipos de mensaje:

- *Mensaje modo normal:* Es el utilizado en la comunicación normal. Es la trama EIB cifrada con AES 128. Las cabeceras tanto de la LPDU como de la NPDU se excluyen del cifrado El mensaje normal que tenemos es este:

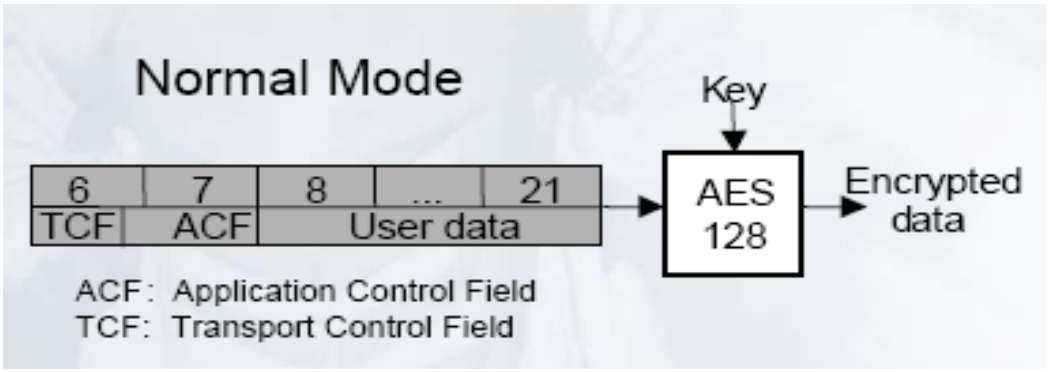


Figura 33. Mensaje modo normal

- *Mensaje modo contador:* Se utiliza en el modo contador y se construye a partir de la trama EIB. Se excluyen las cabeceras de la LPDU y NPDU, calculándose en los demás datos un *Cyclic Redudancy Code 32 bits (CRC32)* que se añadirá al final. Al resultado se hace una operación de XOR con un contador de 128 bits que tiene almacenado en memoria el dispositivo. Por último, se hace el cifrado de todo lo anterior menos las cabeceras.

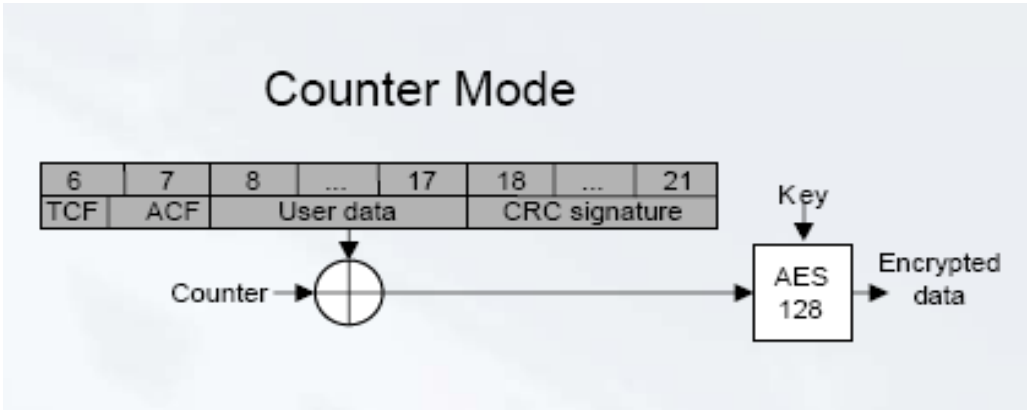


Figura 34. Mensaje modo contador

2.13.4 Tipos de comunicación

Hay dos tipos diferentes de comunicación: gestión de la comunicación y la comunicación de grupo.

- *Gestión de la comunicación:* son tareas de configuración y mantenimiento. Los servicios de gestión se invocan en los dispositivos destino, creándose sesiones punto a punto.

- *Comunicación de grupo:* en la que los datos del proceso se intercambian en grupos de comunicación exclusivamente. Varios remitentes son capaces de enviar los datos del proceso a varios receptores de acuerdo a unos identificadores de mensajes donde los remitentes y los receptores no son conscientes el uno del otro.

2.13.5 Funcionamiento de EIBSec

Para entender un poco el funcionamiento de EIBSec, vamos a ver más detalladamente el proceso de intercambio de mensajes en los dos tipos de comunicación ya vistos.

2.13.5.1 Gestión de la Comunicación

Como ya se ha mencionado, se realizan tareas de configuración y mantenimiento por lo que una sesión tiene que ser establecida para permitir un intercambio seguro de datos de gestión. Este establecimiento de sesión se basa en una variante segura del protocolo de Needham-Schroeder, que es un protocolo de comunicaciones usado sobre una red insegura. Este protocolo tiene dos variantes en las que se base EIBSec que son clave simétrica y clave pública.

Si un dispositivo quiere establecer una sesión con otro dispositivo, previamente el que inicia la sesión envía un mensaje al ACU, Session Key Request, donde especifica la dirección del dispositivo con el que quiere establecer la sesión. El ACU, envía los mensajes de inicio de conexión al otro dispositivo incluyendo parte de los campos recibidos en el anterior mensaje. Posteriormente, el ACU genera una clave de 128 bits que se distribuye a ambos dispositivos en dos mensajes. Una vez que los dispositivos tienen la clave base de sesión, son capaces de calcular la clave y el contador de sesión para realizar el cifrado de los mensajes que se intercambiarán.

2.13.5.2 Comunicación de grupo

Para proteger la comunicación de grupo frente a ataques de seguridad, todos los mensajes de grupo se cifran en el modo contador. En este tipo de comunicación no es necesario distribuir las claves de grupo durante la instalación. Si un dispositivo quiere unirse a un grupo de comunicación, puede obtener la clave de grupo y el contador del ACU correspondiente.

Si un dispositivo quiere unirse a un grupo envía un mensaje al ACU, *Join Group Request,* donde especifica el grupo al que se quiere unir. El ACU, después de comprobar si el dispositivo está autorizado a unirse al grupo, le envía la clave de grupo y el contador. A partir de aquí podrá cifrar y descifrar los mensajes del grupo en modo contador.

Capítulo 3: INTRODUCCIÓN A OMNeT++

3.1 INTRODUCCIÓN

La historia del desarrollo de *Objective Modular Network Testbed in C++ (OMNeT++)* no es reciente. Su autor, Andras Varga, empezó a desarrollar este sistema en 1992 a partir del simulador Omnet escrito en Pascal y desarrollado por el Dr Gyrgy Pongor en la Universidad Técnica de Budapest **[10].**

OMNeT++ es un simulador de eventos discretos modular, orientado a objetos **[11]**. Un modelo en OMNeT++ consiste en módulos jerárquicamente anidados, que se comunican mediante paso de mensajes. Presenta dos interfaces de ejecución: una grafica y otra de comandos. El modo grafico permite una simulación por pasos o de forma continua. La interfaz visual, además, es una herramienta didáctica y de depuración fundamental.

OMNeT dispone de varios modelos para redes cableadas e inalámbricas y otros nuevos algoritmos de encaminamientos de red se están incorporando con parte del resultado de proyectos de investigación. El entorno de simulación no se limita a los modelos de red, ya que puede simular una amplia gama de soluciones como pueden ser canalización de modelos, procesos de comunicación, redes basadas en colas o arquitecturas de hardware.

OMNET++ proporciona una arquitectura de componentes por modelos. Los modelos están compuestos por componentes reutilizables módulos, que se pueden combinar de varias maneras. Los componentes o módulos se programan en C++, y son posteriormente ensamblados usando un lenguaje de alto nivel llamado NED. Los módulos pueden conectarse entre sí a través de puertas y combinarse para formar módulos compuestos. La profundidad de anidación del módulo no está limitada. Los módulos se comunican a través de mensajes, los cuales llevan las estructuras de datos. Los módulos pueden llevar los mensajes a través de la red por medio de las puertas (*gates)* y las conexiones. En el caso de simulaciones inalámbricas, los mensajes son enviados directamente a su destino. Los módulos pueden contener parámetros que son utilizados para personalizar el comportamiento del módulo y / o parametrizar la topología del modelo.

3.2 CONCEPTOS

Un modelo OMNeT ++ está compuesto de módulos jerarquizados los cuales se comunican por medio de intercambio de mensajes a través de puertas

que conectan estos módulos entre sí. Hay dos tipos de módulos: simples y compuestos.

El módulo de nivel superior llamado Módulo de sistema, contiene submódulos que a su vez pueden contener otros submódulos. No hay limitación en el nivel de anidación de módulos (llamados módulos compuestos) que hacen posible la construcción del modelo de simulación de acuerdo a la estructura lógica del modelo real considerado.

Los módulos simples son los componentes activos en el modelo. Son programados en C++, usando la librería de clases de OMNeT++. En estos módulos se implementa el comportamiento del modelo simulado. Pueden formar directamente la red o estar contenidos dentro de otros módulos, denominados módulos compuestos, que junto a otros módulos simples determinan el funcionamiento global del modulo compuesto.

En la librería de clases de OMNeT++ existen una clase básica llamada cSimpleModule, que es la base de todos los módulos simples. Esta clase tiene definidos una serie de funciones virtuales, que por sí solas no realizan ninguna funcionalidad específica, y tienen que ser redefinidas en los módulos simples para realizar el comportamiento requerido en la simulación. Como mínimo hay dos funciones que se vuelven a definir dentro del modulo:

- *initialize():* inicializa el modulo, leyendo los parámetros definidos en el fichero NED que determinan la funcionalidad del objeto.

- *handleMessage():* es el método que se llama cada vez que el módulo recibe un evento de alguna de las conexiones de entrada. Esta función se implementa, ya que es virtual, dependiendo del comportamiento requerido para el módulo simple donde está definida.

Opcionalmente existe otro método que puede ser implementado, el método *finish(),* que es llamado cuando finaliza la simulación de manera correcta. Este método puede contener la grabación de valores estadísticos de la red, con el método *recordScalar(),* que se hallan calculado durante la simulación, así como ficheros de log que son finalizados y cerrados para su posterior lectura.

Los componentes o *módulo*s se conectan con otros módulos a través de las puertas de enlace llamadas "gates", que son las interfaces entre los módulos y las conexiones. Los módulos pueden estar conectados con módulos

de su mismo nivel jerárquico dentro de un modulo compuesto o directamente al modulo compuesto.

Las gates son los interfaces de entrada y salida de los módulos. Los mensajes se envían a través de puertas de salida y llegar a través de puertas de entrada. Una puerta es como una tubería unidireccional para mensajes por lo que cuando tenemos que diseñar una comunicación bidireccional entre módulos. Es necesario definir las puertas y conexiones en cada dirección.

Cada *conexión* (también llamada enlace), se crea en un solo nivel de la jerarquía. Puede conectar las puertas correspondientes de dos submódulos, o una puerta de un submódulo y otra del módulo compuesto.

Las conexiones pueden tener opcionalmente parámetros para especificar su velocidad, tasa de error y retardo de propagación. Los parámetros se usan para calcular el retardo de los mensajes en la simulación y las estadísticas de errores (paquetes transmitidos).

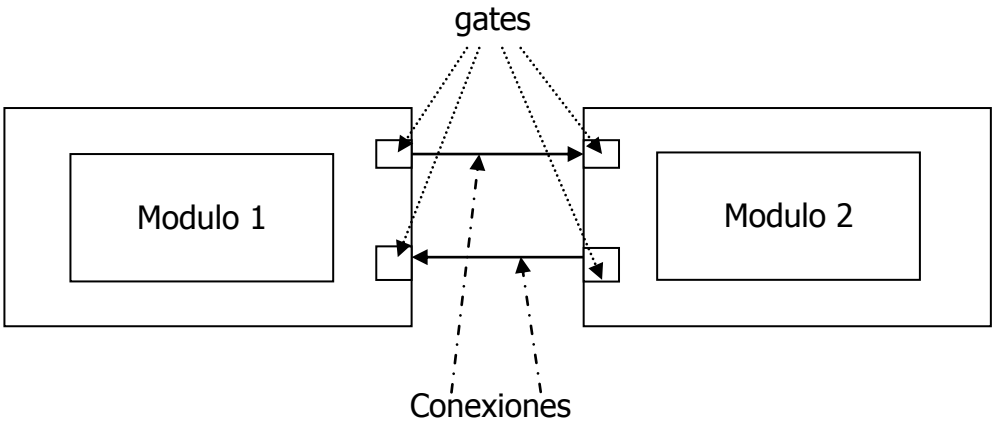


Figura 35 Módulo con puertas de enlace y conexiones

El módulo de parámetros es usado para pasar información específica a los objetos individuales de cada modulo y estos son definidos en el fichero NED o en un fichero de configuración (.ini). Los valores de los parámetros pueden ser numéricos o cadenas de caracteres.

El fichero NED es un fichero de definición de red para modelo de simulación OMNeT++. Éste contiene las definiciones tanto de los módulos simples como de los módulos compuestos, las definiciones de las puertas de enlace de un modulo y las conexiones entre módulos.

La funcionalidad de los módulos simples se implementa en C++. OMNet++ dispone de una librería de clases cuya funcionalidad soporta la flexibilidad del lenguaje C++. La librería OMNeT++ contiene, por ejemplo, los objetos simulados en los módulos, puertas de enlace, conexiones, parámetros, mensajes, clases contenedoras y clases con colecciones de datos. Se pueden crear dinámicamente tanto módulos como mensajes.

3.3. EL LENGUAJE NED

Para hacer uso de todo lo visto en el apartado anterior, hace falta describir un modelo de simulación que haga uso de los módulos. Es necesario que el usuario defina la estructura del modelo de simulación. Para ello, se hace uso del lenguaje NED. Éste lenguaje describe la relación entre los módulos simples, módulos compuesto, puertas, conexiones y módulos de parámetros.

A continuación mostramos un ejemplo de un fichero NED que es parte de la simulación realizad en el presente proyecto. Como podemos apreciar nos muestra la definición de un dispositivo de la red, parámetros, puertas, submódulos y conexiones:

```
module KNXEib_device
{
    parameters:
        string individualAddress;
        string groupAddress;
        string applicationType;
        bool actuator;
        @display("b=90,170;i=misc/switchOff");
    gates:
        input in_bus;
        input in_user;
        output out_bus;
    submodules:
        protocol: KNXEib_protocol {
            parameters:
                individualAddress = individualAddress;
                groupAddress = groupAddress;
                applicationType = applicationType;
                @display("p=55,125;i=block/layer,cyan");
        }

        application: KNXEib_application {
            parameters:
                actuator = actuator;
                applicationType = applicationType;
                groupAddress = groupAddress;
                @display("p=55,55;i=block/process,grey");
        }

    connections:
        //conexiones externas
        out_bus <-- { @display("m=s"); } <-- protocol.out_bus;
        in_bus --> { @display("m=s"); } --> protocol.in_bus;
        in_user --> application.in_user;
        //conexiones internas
```

```
        protocol.out_application --> { @display("m=n"); } -->
application.in_application;
        protocol.in_application <-- { @display("m=n"); } <--
application.out_application;

      }// end KNXEib_device
```

La sección de parámetros define los parámetros necesarios para que el dispositivo opere. Estos parámetros son: una dirección hardware, una tabla de aplicaciones asociadas y el tipo de aplicación.

La sección de puertas (*gates)* define cómo el dispositivo se comunica con otros objetos y, por tanto, representa la interfaz del dispositivo.

La sección de submódulos define que objetos componen el módulo, aparte de ser parametrizables.

La sección de conexiones define que puertas son de entrada o de salida y donde están conectadas, especialmente en lo que se refiere a los submódulos utilizados.

3.4 EJECUCIÓN DE UNA SIMULACIÓN OMNeT++

El sistema de simulación de OMNET++ nos proporciona un núcleo de simulación que contiene el código que gestiona la simulación y la librería de clases C++ de OMNET, así como un interfaz de usuario que nos permite interaccionar con los módulos definidos y visualizar la simulación del modelo.

La simulación consiste en:

- *Fichero NED (.ned)*: contiene definiciones de módulos simples, módulos complejos, puertas y conexiones describiendo la estructura de la simulación.
- *Fichero de definición de mensajes (.msg):* es opcional. Puede contener varios tipos de datos para añadir a los campos de los mensajes que intercambian los módulos. Los ficheros de definición de mensajes son traducidos a clases C++ e integrados dentro de la simulación.
- *Ficheros de módulos simples:* en código fuente C++ (.h y .cpp) que implementan funcionalidad de los módulos.

3.5 DATOS Y REPRESENTACIÓN DE LOS RESULTADOS DE SIMULACIÓN.

Tenemos dos tipos de datos diferentes en la simulación: uno es un colección de datos almacenados en un vector y otro es un solo dato almacenado en un escalar.

Los vectores utilizados, llamados *output vector*, son una serie de valores almacenados durante la simulación junto con el instante de tiempo en el que se almacena el valor. Se pueden utilizar para obtener distintas medidas durante la simulación. En nuestro caso se utiliza para medir la carga de cada línea en ciertos instantes de tiempo así como el tiempo en el que la línea está ocupada transmitiendo y no puede transmitir otros paquetes.

Para almacenar los valores deseados en los vectores durante la simulación, simplemente utilizaremos los métodos del modulo simple cOutVector.h y se almacenan en un fichero con extensión .vec. Éstos contienen dos tipos de declaración: vectores y datos.

Las líneas de declaración de los vectores son los siguientes:

```
vector 6  KNXEib_area_1.main_line.bus_1_0  bus_1_0-vectorLoad  ETV

vector 7  KNXEib_area_1.main_line.bus_1_0  bus_1_0-vectorBusy  ETV
```

donde se nos muestra "vector", el numero identificativo del vector, el modulo que lo ha creado (path completo), nombre del vector (que pueden ser dos: el de carga de la red y el de ocupación de la red) y la multiplicidad.

Un ejemplo de línea de declaración de datos es la siguiente:

```
6     186   0     0.020384
```

donde nos muestra el numero identificativo del vector, el numero de evento que se ha almacenado, el instante de tiempo de la simulación y el dato correspondiente a ese instante de tiempo.

En cuanto a las valores escalares estadísticos, son una manera de comparar el comportamiento del modelo según los valores que hayamos calculado y almacenado. Hay un método que se invoca al finalizar la simulación, con el método de modulo simple *finish(),* llamado *recordScalar()* al que se le pasan como parámetro el nombre que se le quiere dar al valor junto con el valor que queramos almacenar. Un ejemplo del formato de los parámetros del bus recogidos durante una simulación del proyecto son los siguientes:

```
scalar KNXEib_area_1.main_line.bus_1_0 "Bus Ocupado"       9

scalar KNXEib_area_1.main_line.bus_1_0 "Mensajes enviados"       22

scalar KNXEib_area_1.main_line.bus_1_0 "Mensajes recibidos"      31

scalar KNXEib_area_1.main_line.bus_1_0 "Tiempo          de          simulación"
       12.024620364301

scalar KNXEib_area_1.main_line.bus_1_0 "Bytes recibidos"   279

scalar KNXEib_area_1.main_line.bus_1_0 "Bytes enviados"    198
```

A partir de estos datos obtenidos durante la simulación podemos utilizar las herramientas proporcionadas por OMNeT++ para el análisis y representación de los datos.

Capítulo 4: DISEÑO DEL SIMULADOR

El simulador ha sido diseñado en módulos, según las especificaciones del protocolo EIB y las de OMNeT++.

Existen dos configuraciones de simulación distintas, que van a tener elementos comunes y otros que son específicos de cada una de ellas.

La organización del proyecto está compuesta por: redes simuladas, clases implementadas, mensajes y órdenes de ejecución de la red.

4.1 REDES

Hay dos redes distintas implementadas: red EIB y red EIBSec. Esta la componen tres ficheros .NED, donde están definidas las redes que vamos a utilizar en la simulación así como la conexión entre los diferentes módulos. Los tres ficheros son:

- *PFC_KNXEIBSec_basicModules.ned*: donde están los módulos simples de la red implementados.
- *PFC_KNXEibSec_onlyEIB.ned*: donde está la red EIB implementada, y son módulos compuestos cuyos submódulos son los módulos simples.
- *PFC_KNXEibSec.ned*: donde esta implementada la red EIB con seguridad, y está compuesto igualmente de módulos compuestos.

Inicialmente, se ha descartado el uso de backbone line y de backbone coupler por simplicidad de la red implementada. Se podría incluir el uso de estos backbones (línea y acoplador) ya que su comportamiento es el mismo que el de la línea principal y el acoplador de línea utilizados en las simulaciones.

4.2 CLASES

Se han diseñado tantas clases como módulos simples componen la red. Cada uno de ellos define la funcionalidad y comportamiento de cada componente. Además, se ha definido una clase en C++, que tiene implementadas funciones específicas para los mensajes EIB, tanto para los de EIB como los de EIBSec.

Dentro de las clases hay dos tipos: comunes y específicos de la red. En el diseño se ha optado por dejar en las clases comunes los componentes que

tiene un comportamiento igual o similar para ambos modos de simulación. Hay otros componentes, cuyo comportamiento es diferente para ambas redes simuladas, que son específicos. Esta decisión se ha tomado para simplificar los módulos y el tiempo de procesado del simulador.

4.2.1 Clases comunes

Las clases comunes, como su nombre indica, son las que tienen un comportamiento idéntico a ambas redes sin importar el tipo de red que se va a ejecutar. Estas clases son las siguientes:

- *userSimulator:* es la clase que simula al usuario en una red real.
- *KNXEib_application:* es el módulo de aplicación dentro del dispositivo.
- *KNXEib_bus:* simula el comportamiento del bus dentro de una red real.
- *KNXEib_messageFunctions:* clase auxiliar diseñada en C++, para la creación de los mensajes EIB. Esta clase es utilizada por el módulo de protocolo tanto para generar la LPDU que se envía al Bus como extraer los datos del mensaje para enviarlos al módulo de aplicación.

4.2.2 Clases específicas

Se han diseñado una estructura de clases para hacer diferenciación de ciertos componentes dependiendo de la red que se simule. Se ha optado por este diseño para resultar un código más sencillo y eficiente. Tienen una base común que parte del funcionamiento del protocolo EIB, añadiéndose unas modificaciones para el caso de la red EIBSec.

- EIB
    - *KNXEib_protocol:* clase que simula el funcionamiento del protocolo EIB sin seguridad habilitada.
    - KNXEib_deviceCoupler: hace la función de acoplador de línea en la red, en la que filtra los mensajes de su línea.
- EIBSec
    - *KNXEibSec_protocol:* funcionamiento del protocolo EIB con la variante de seguridad añadida. Aparte de componen la trama realiza las funciones de cifrado y descifrado de los mensajes

- KNXEibSec_deviceACU: realiza las mismas funciones que el acoplador de línea, con algunos añadidos como generación de claves y contador así como el envío de estos en mensajes seguros.

4.3 MENSAJES

Tipos de mensajes, cuyo fichero tiene la extensión .msg, que se intercambian entre los distintos módulos. Hay varios tipos de mensajes:

- *applicationMessage:* mensaje de tipo aplicación, que intercambia el componente de aplicación entre los módulos a los que está conectado. Es de tipo común, ya que es el mismo para ambas redes.

- *KNXEib_Frame:* mensaje EIB que contiene los caracteres que especificados en la documentación técnica del protocolo. Especifico para la red EIB.

- *KNXEibSec_message:* mensaje EIBSec, específico para este modo de simulación, que contiene la trama tal cual se nos define en la documentación técnica de EIBSec.

- *securityMessage:* mensaje de seguridad intercambiados entre el ACU y el protocolo para especificar la clave y el contador con el que se van a cifrar los mensajes pertenecientes a EIBSec.

4.4. ORDENES DE EJECUCIÓN

Es un fichero *txt*, del cual lee uno de los módulos simples implementados en la red, que contiene las ordenes a ejecutar durante la simulación. Este fichero es el que nos define o simula el comportamiento del usuario hacia la red.

Una línea de este fichero, que nos muestra un ejemplo de las órdenes que se ejecutan, es:

```
0    0.001    10   switch1   switchingAll   ON
```

donde se específica el instante de tiempo en el que sucede el evento en la simulación, el intervalo de tiempo entre ellos, el numero de eventos que se van a crear en la simulación, el nombre de la dirección a la que dirigido y los datos

que se quieren ejecutar en el módulo de aplicación del dispositivo al que va dirigido.

Este fichero se pasa como parámetro a uno de los módulos diseñados, el que simula al usuario, que lo lee e interpreta en un mensaje para posteriormente enviarlo al dispositivo correspondiente.

4.5 ESTADÍSTICAS

La parte estadística se recoge en el modulo del Bus de cada línea, que es donde realmente se mide la carga de la red. Como ya se menciono anteriormente en el capítulo anterior, las estadísticas se recogen en los vectores de salida.

En nuestro caso tendremos dos vectores por cada bus (contenido dentro de una línea): *nombreBus-vectorLoad* y *nombreBus-vectorBusy*. Estos almacenan el valor de carga (*load)* y el número de veces que el bus está ocupado (*busy)* cada segundo. Un ejemplo de formato de este vector recogido de una de las simulaciones es el siguiente:

```
vector 6  KNXEib_area_1.main_line.bus_1_0  bus_1_0-vectorLoad  ETV

vector 7  KNXEib_area_1.main_line.bus_1_0  bus_1_0-vectorBusy  ETV

6      186    0      0.020384

6      336    1      0.020384

6      564    2      0.040768

6      792    3      0.040768

6      1020   4      0.040768

6      1248   5      0.040768

6      1476   6      0.040768

6      1704   7      0.040768

6      1932   8      0.040768

6      2160   9      0.040768

6      2350   10     0.040768

6      2462   11     0.020384

7      186    0      0

7      336    1      0

7      564    2      1
```

```
7	792	3	1
7	1020	4	1
7	1248	5	1
7	1476	6	1
7	1704	7	1
7	1932	8	1
7	2160	9	1
7	2350	10	1
7	2462	11	0
```

Como podemos ver hay dos tipos de líneas: las líneas de declaración de los vectores y las de datos. En la declaración, el contenido de ésta es el número de vector, el nombre completo del módulo donde se recogen los vectores y su nombre respectivamente. En el caso de lo datos, tendremos el número del vector, el número de evento que ha generado ese dato, el instante de la simulación en el que se genera y el dato estadístico.

Capítulo 5: DESARROLLO DE MÓDULOS OMNeT++

En la figura 34, se muestra un resumen a alto nivel del modelo de red implementada para ambos casos de simulación. Se procede de esta manera dado que la mayoría de los módulos son comunes para ambos modos de simulación. El modelo de red implementado gráficamente es igual en ambos casos, difiriendo únicamente en el comportamiento del acoplador y del protocolo. Esta diferenciación se explica más adelante cuando se define exhaustivamente cada modulo.

La línea es un módulo compuesto que está formada por otros módulos simples. Hay distintos tipos de líneas que se diferencian en su comportamiento dependiendo del modulo simple o compuesto que contenga. En el caso de la línea principal, tan solo está compuesta por un modulo simple que es el bus. Sus tareas serán envío y recepción de los datos. La otra clase de línea, aparte de contener el bus, contiene varios módulos compuestos que son los dispositivos y un modulo simple que es el que hace la funciones del usuario.

Los dispositivos, contenidos en la línea, son módulos compuestos formados por dos módulos simples. Estos dos módulos simples se han creado para dividir las tareas que hace el dispositivo. Una de ellas es la creación, envío y recepción de los mensajes de la red, y la otra es la de ejecución de las órdenes recibidas en los mensajes. Estos módulos han sido llamados protocolo y aplicación respectivamente.

Debido a que es una simulación y no existe un usuario real que interaccione con la red, se ha diseñado un modulo que realiza estas funciones. Está contenido dentro de la línea, conectado a todos los dispositivos de ésta. Este modulo es quién realiza la acción del usuario y hace que la red sea simulada. La ordenes se leen de un fichero, son interpretadas y enviadas al modulo de aplicación de todos los dispositivos. El modulo de aplicación del cada dispositivo mira el contenido del mensaje para verificar si la orden recibida va dirigida a él. La parte de dispositivo que realiza el funcionamiento del protocolo recibirá un mensaje en el caso de que sea el dispositivo que ejecuta la orden del usuario.

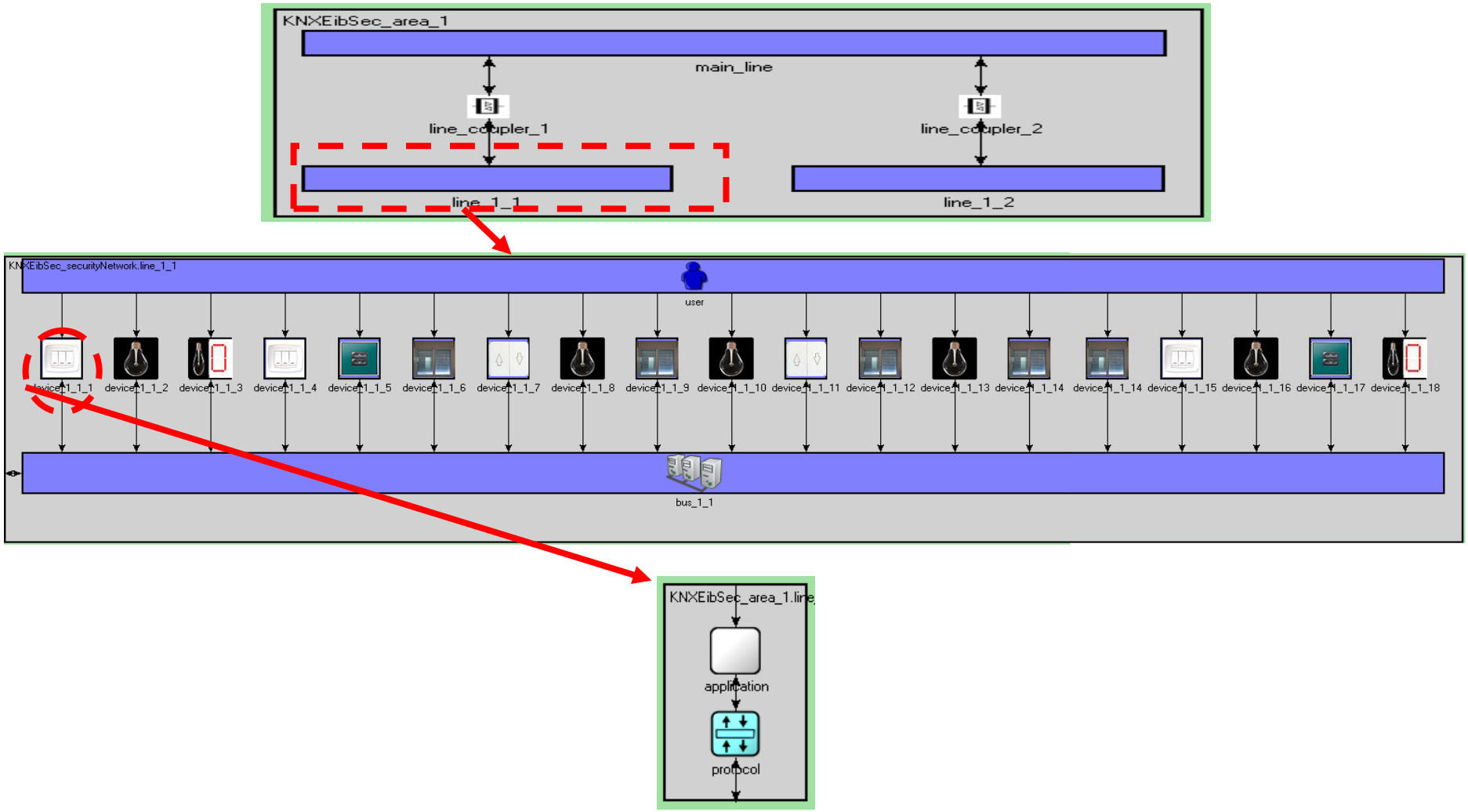


Figura 36 Modelo de red

5.1 OBJETOS C++

Los objetos C++ a los que hacemos referencia son los objetos que se intercambian entre módulos. OMNeT++ nos proporciona un tipo de objetos llamados mensajes. Los mensajes se definen según las necesidades y su contenido es la información que se trasmite entre módulos.

Estos objetos o mensajes van encapsulados en un mensaje OMNeT++. Antes de la compilación estos mensajes son procesados por el compilador *opp_mgsc* que genera el fichero C++. Este fichero a su vez es procesado de nuevo para alinear los tipos de datos que se han utilizado, dándonos como resultado un fichero .cc y otro .h que es código en c utilizado para la compilación global del proyecto **[11].**

El fichero resultado del procesado del mensaje creado, es la base para definir la estructura del objeto embebido *cMessage* en OMNeT++. El *cMessage* integra datos binarios de objetos con el método *encapsulate()* y recupera de cada el tipo de mensaje original con el método *decapsulate()* de OMNeT++.

Tenemos varios tipos de mensajes que son:

- *applicationMessage:* mensaje de aplicación intercambiado entre los módulos usuario, aplicación y protocolo.
- *KNXEib_Frame:* mensaje EIB cuyo contenido es la LPDU a enviar, dividida en octetos añadiéndoles los bits de paridad.
- *securityMessage:* mensajes de seguridad intercambiados entre el ACU y el dispositivo para enviar tanto la clave de seguridad como el contador.
- *KNXEibSec_message:* mensaje EIBSec cuyo contenido es el mensaje en modo contador enviado a través de la red segura.

5.1.1 Mensaje aplicación

Este mensaje es el que se ha llamado en la simulación *applicationMessage.* Contiene la información que intercambia a nivel interno el dispositivo. Se ha definido este mensaje como la comunicación entre los módulos de protocolo-aplicación y aplicación-usuario.

Este mensaje no está definido en el estándar EIB, sino que se ha definido como una forma de comunicación entre los módulos internos que forman el dispositivo. Se intercambia entre el usuario, aplicación y protocolo.

ApplicationMessage es común a ambos modos de simulación y su contenido es el siguiente:

```
message applicationMessage
{
    string application;
    string address;
    string data;

}
```

Los campos de la estructura corresponden a: *application* es el nombre de la aplicación, *address* es la dirección de grupo o individual a la que va destinado el mensaje y *data* es la orden que da el usuario para ejecutar en la red.

5.1.2 Mensaje KNX EIB

Es el llamado *KNXEib_Frame.* Es la trama EIB llamada LPDU, especificada en el estándar EIB. Es el mensaje intercambiado entre todos los dispositivos de la red simulada. Lo crea el módulo de protocolo, perteneciente al dispositivo, y lo envía hacia el bus para que viaje a través de la red. Una vez llega a su destino, de nuevo el módulo protocolo perteneciente al dispositivo destino, lo chequea y analiza para verificar la información. Invocando a las funciones auxiliares de *KNXEib_messageFunctions,* va extrayendo las PDUs de las distintas capas (NDPU, TPDU y APDU). Una vez extraídas, comprueba los campos y obtiene los datos que serán la ejecución a realizar.

El contenido de la LPDU es el siguiente:

```
message KNXEib_Frame
{
    string messageFrame[23];
    unsigned int totalMessageLenght;

}
```

Los campos son: *messageFrame* es la trama completa dividida en caracteres como lo hace el protocolo EIB y *totalMessageLenght* es la longitud total del mensaje que enviamos.

5.1.3 Mensaje de seguridad EIB

Son los mensajes que se distribuye en la red EIB con seguridad o EIBSec, llamados *securityMessage.msg.* Estos mensajes son los que manda el acoplador de línea, cuando ha recibido un mensaje del dispositivo para unirse a un grupo. El acoplador envía estos mensajes, para informar de la clave y el

contador con los que se va a cifrar el mensaje antes de ser transmitido por la red.

El contenido del mensaje es el siguiente:

```
message securityMessage
{
    char data [16];

}
```

Es el mismo mensaje para enviar tanto la clave como el contador. El campo que lo compone es uno solo llamado *data.* Es un array de 16 caracteres cuyo contenido puede ser la clave o el contador, dependiendo del nombre del mensaje enviado.

Cuando el dispositivo recibe el mensaje de seguridad sabe cual es su contenido dependiendo del nombre del mensaje. Son los llamados *Join Group Responde* y *Group Resync Response* los que contiene la clave y el contador respectivamente.

5.1.4 Mensaje KNX EIBSec

Este mensaje contiene la trama EIB con ciertas modificaciones ya que se cifran ciertos campos de la trama original. El contenido del mensaje *KNXEibSec_message.msg* es el siguiente:

```
message KNXEibSec_message
{
    string controlField;
    string addressSource;
    string addressReceiver;
    string routingCounter;
    string length;
    string encryptedData;
    unsigned int totalMessageLenght; //longitud de los datos cifrados
    unsigned int lengthData; //longitud de los datos
    unsigned int lengthDataBeforeCipher; //longitud del mensaje resultado de
la xor

}
```

Los campos contenidos en este mensaje son: *controlField, addressSource, addressReceiver, routingCounter y length* son los primeros campos de la trama EIB original, *encryptedData* son los demás campos de la trama que son cifrados, *totalMessageLength* es la longitud total de los datos cifrados, *lengthData* es la longitud de los datos sin cifrar y *lengthDataBeforecipher* es la longitud de los datos después de hacer XOR con el contador para después hacer el cifrado.

5.2 MÓDULOS

Como comentamos en la sección 3.2, los módulos simples son los que definen los componentes activos de la red. En la simulación del presente proyecto se han definido siete módulos simples, que son básicos para el funcionamiento especificado. Los módulos compuestos utilizados, contenedores de módulos simples, son diseñados tal y como sería en una red real.

5.2.1 Usuario simulado – userSimulator

El módulo userSimulator no representa ninguna función específica de EIB. El propósito de este modulo es simular la interacción del usuario con los dispositivos EIB definidos en la red. Simula eventos que son creados al principio de la simulación y define cuando finaliza la simulación ya que no existen más eventos en la cola para ser procesados por OMNeT++.

El userSimulator en el fichero NED es el siguiente:

```
// Modulo simple userSimulator
// Modulo que simula un usuario cuando interactúa con los elementos de
la red EIB, mandando mensaje a la aplicación.
// Parameters:
// No tiene
//
simple userSimulator
{
    parameters:
        string userSimulatorFile = "UserSimulator.txt";
        @display("i=abstract/person");
    gates:
        output out[];


}// end userSimulator
```

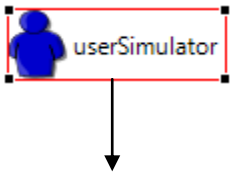


Figura 37 userSimulator

En este módulo, solo se define la función virtual initialize(). Aparte de inicializar el módulo, parsea el fichero que contiene las ordenes del usuario. Una vez obtenidas las órdenes de ejecución, crea eventos y los programa en el tiempo estipulado para que viaje a lo largo de la red.

Este documento contiene órdenes como las siguientes:

```
0       0.001  10     switch1       switchingAll ON

2       0.002  10     switch4       switchingAll OFF
```

Los parámetros contenidos en este fichero son los siguientes:

- *Tiempo:* instante de tiempo que queremos se envíe la orden.
- *Intervalo Tiempo*: cada cuanto tiempo se envía esa orden.
- *Contador:* número de veces que se envía.
- *Aplicación:* nombre de la aplicación (cada una tiene su nombre de aplicación definida en el fichero NED) que ejecuta la orden.
- *Dirección:* nombre del grupo a la que se dirige la orden.
- *Dato:* la orden que se manda ejecutar en el dispositivo.

Por ejemplo, la siguiente orden

```
0       0.001  10     switch1       switchingAll ON
```

se interpreta como: en el instante de tiempo 0 se van a enviar 10 mensajes con un intervalo de tiempo de 0,001 segundos entre ellos. El mensaje se va a enviar desde el switch1, que es un interruptor. La orden que se envía a través de este interruptor es encender todas las luces y abrir todas las persianas (switchingAll es el grupo 0/0/0 que corresponde a todos los dispositivos conectados: persianas y luces sin contar las luces de regulación). Esto es interpretado por el módulo userSimulator, traduciéndolo en un mensaje *applicationMessage* que será enviado a través de sus puertas de salida hacia el módulo de aplicación.

La programación de eventos en la cola de salida se calcula en base a los parámetros tiempo e intervalo de tiempo. El instante de tiempo en el que se envía el mensaje sigue una distribución uniforme, cuyos límites son calculados a partir de los tiempos ya menciones. El número de veces que se hacen estos cálculos y se programa el envío del mensaje, es el indicado por el contador. Esta descripción de la programación de los eventos la podemos ver en el siguiente extracto de código:

```
for (int i = 0; i <= counter; i++)
{
	double limit_low = 0.0f;
	double limit_high = 0.0f;
	double timeStamp = 0.0f;

	//Calcula el limite superior y el limite inferior de la uniforme
	limit_low = scheduleTime + ((double) i * interval) - (0.1 * interval);
	limit_high = scheduleTime + ((double) i * interval) + (0.1 * interval);

	if (limit_low < 0.0f) limit_low = 0.0f;
	timeStamp = uniform(limit_low, limit_high);
	applicationMessage *msgApplication = generateMessage();
	for (int i =0; i<this->gateSize("out"); i++)
	{
		applicationMessage *msgCopy;
		msgCopy = (applicationMessage *) msgApplication->dup();
		sendDelayed(msgCopy, timeStamp, "out", i);
		bubble("Enviada la orden al device");
	}

}
```

El programador de eventos calcula el instante de tiempo en la que se va a enviar el mensaje. Este cálculo es el valor central de la uniforme que está delimitada por dos valores, superior e inferior, calculados previamente en base a los tiempos indicados en fichero de usuario. Gráficamente la programación de eventos es la siguiente:

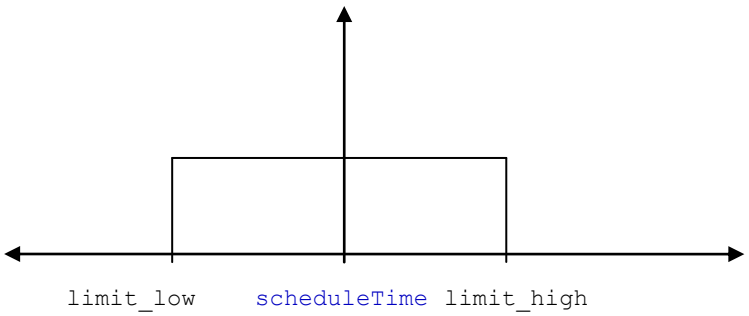


Figura 38 Programación de Eventos

Únicamente hemos hablado de una programación de eventos, que es llamada desde la función que inicializa el módulo y lee el fichero de usuario. Este modulo, es el único en el que no se redefine la función *handleMessage* como bien se comentaba en la sección 3.2. Este método es únicamente implementado en el caso de módulos que reciban eventos. El módulo userSimulator únicamente tiene puertas de salida, ya que el usuario envía órdenes pero no las recibe.

Las puertas de salida estarán conectadas a cada dispositivo de la red. No se definen número de puertas, sino que dependerá del número de dispositivos, por lo que se define un array de puertas de salida y se va incrementando la

puerta conectado con el operador ++ como se muestra a continuación en el ejemplo siguiente:

```
device_1_1_1.in_user <-- user.out++;
device_1_1_2.in_user <-- user.out++;
device_1_1_3.in_user <-- user.out++;
device_1_1_4.in_user <-- user.out++;
```

5.2.2 Aplicación KNX/EIB – KNXEib_application

KNXEib_application es otro módulo simple de la red y es uno de los que componen el dispositivo (módulo compuesto).

Se comunica con el módulo *KNXEib_protocol,* que se define en la siguiente sección, a través de los mensajes de tipo *applicationMessage* definidos en la sección 5.1. Como comentamos anteriormente, este tipo de mensaje es una forma de comunicación que se ha diseñado en el proyecto entre los módulos que no se corresponden con ningún componente real EIB.

Cuando llega al módulo un mensaje procedente del usuario, verifica si es para él mirando el campo *application*, y si es así, obtiene la dirección del mensaje. Esta dirección la comprueba en su tabla de direcciones verificando si la contiene. Si es contenida, cambia el nombre del grupo por la dirección de grupo en el mensaje para su posterior envío al modulo de protocolo.

El modulo *KNXEib_application* definido en el fichero NED es:

```
// Modulo simple KNXEib_application
// Implementa una aplicación en EIB como actuadores o sensores.
// Parameters;
//     string applicationType: tipo de aplicación EIB.
//     string groupAddress: Dirección o direcciones de grupo a las que
       pertenece la aplicación.


simple KNXEib_application
{
    parameters:
        bool actuator;
        string applicationType;
        string groupAddress;
         string individualAddressActuators;

    gates:
        input in_application;
        input in_user;
        output out_application;

}// end KNXEib_application
```

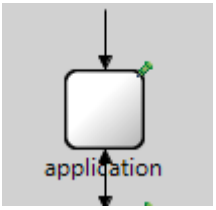


Figura 39 KNXEib_application

Los parámetros que deben especificarse cuando la red es creada son:

- *actuator*: indica si la aplicación es un interruptor o regulador que envía las órdenes o un actuador que ejecuta la órdenes recibidas.

- *applicationType:* nombre de la aplicación. Esta se compara con el nombre de la aplicación que le llega en el mensaje de aplicación procedente del usuario. Así se determina si es el mensaje que ejecutar la orden del usuario especificada en el mensaje.

- *groupAddress:* tabla de direcciones de grupo del dispositivo. Si bien es un interruptor o regulador, es la tabla de direcciones de grupo a las que puede enviar mensajes porque son los dispositivos a los que esta conectados. Si es un actuador son los grupos a los que pertenece de forma que llegándole un mensaje con una dirección de grupo contenida en su tabla debe realizar la acción contenida en dicho mensaje.

- *individualAddressActuators:* tabla de direcciones individuales a las que está conectado el dispositivo. Este parámetro solo aplica al caso de interruptores o reguladores, que es la tabla de direcciones individuales de los dispositivos que están conectados a él y a los que puede mandar mensajes. Es similar a *groupAddress* pero en vez de direcciones de grupo, con direcciones individuales.

Un ejemplo de definición de los parámetros, extraído del fichero NED de la simulación es el siguiente:

```
groupAddress = "1/1/1=light;1/1/7=light";
applicationType = "light1";
```

Dependiendo del tipo de aplicación (switch, light, switchBlind, blind, dimmer, dimmingLigth) tendrá una representación distinta en la red que se verá reflejada en el dispositivo que lo contiene. Esta parte se ha implementado como mejora visual para entender las acciones que está realizando la red con tan solo ver las imágenes que son representadas. Los diferentes estados de los

dispositivos las podemos a continuación en la siguiente tabla de símbolos de dispositivos:

SIMBOLO ESTADO	TIPO DE APLICACIÓN
	SWITCH OFF
	SWITCH ON
	SWITCH BLIND
	DIMMER

Tabla 8 Símbolos interruptores

En el caso de un interruptor general (luz encendida/apagada) inicialmente su representación será switch off y cambiara a switch on en el momento que este enviando una orden a cualquiera de los dispositivos vinculados a él. En cambio, los interruptores de persiana y regulación se mantendrán igual durante toda la simulación.

Los actuadores pueden tener diferentes estados dependiendo del tipo, como vemos detalladamente en la siguiente tabla de estados:

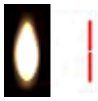
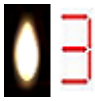
SIMBOLO ESTADO	TIPO ESTADO
	LIGHT OFF
	LIGHT ON
	BLIND CLOSE
	BLIND OPEN
	DIMMING LIGHT OFF MINIMO
	DIMMING LIGHT LEVEL 1
	DIMMING LIGHT LEVEL 2
	DIMMING LIGHT LEVEL 3
	DIMMING LIGHT LEVEL 4 MAXIMO

Tabla 9 Símbolos Actuadores

En el este caso, los actuadores dependiendo de la orden ejecutada tendrá su correspondiente símbolo ayudándonos a visualizar gráficamente su comportamiento. La luz regulada tiene cinco estados diferentes con lo que se

puede representar. Estos cinco estados nos indican el nivel de intensidad de luz que tiene el dispositivo, cuyo nivel máximo es cuatro y el mínimo cero correspondiente al apagado del dispositivo.

5.2.3 Protocolo KNX/EIB – KNXEib_protocol y KNXEibSec_protocol

Protocolo es el encargado de transformar los mensajes procedentes de aplicación, *applicationMessage*, en tramas EIB que van a viajar a lo largo de la red hasta el dispositivo destino o actuador destino. Una vez recibida la trama en el destino ejecuta la acción indicada en ella.

Como se comento en la sección de diseño del simulador, protocolo se ha implementado en dos módulos diferentes. El papel que desempeña dentro de la red es el mismo pero con funcionalidad diferente dependiendo de la red simulada. Si la simulación es la red EIB sin seguridad habilitada se usa *KNXEib_protocol,* mientras que si la red es segura se utiliza *KNXEibSec_protocol.*

La definición del modulo es igual en ambos casos. La diferencia entre ambos modulo es su implementación en C++. Su implementación difiere en la función que maneja los eventos que llegan al módulo y en la forma en la que se construyen los mensajes que viajan a través de la red.

La definición en el fichero NED es:

```
// Modulo simple KNXEib_protocol
// Simula la implementación del protocolo EIB
// Parameters:
// - string individualAddress: Dirección individual del dispositivo
// - string groupAddress: Dirección o direcciones de grupo a las que pertenece
el elemento.
simple KNXEib_protocol
{
    parameters:
        string individualAddress;
        string groupAddress;
        string applicationType;

    gates:
        input in_bus;
        output out_bus;
        input in_application;
        output out_application;

}//end KNXEib_protocol
```

El módulo de protocolo requiere la especificación de tres parámetros:

- *individualAddress:* dirección individual del dispositivo al que pertenece.

- *groupAddress:* tabla de relación de las direcciones de grupo del dispositivo. Esta tabla, contendrá igual su correspondiente modulo de aplicación.

- *applicationType:* indica qué tipo de aplicación es el dispositivo. Este parámetro contiene la misma información que contenía el parámetro del módulo anterior definido con el mismo nombre.

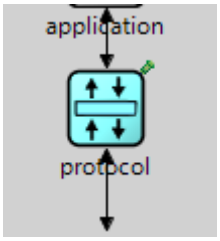


Figura 40 KNXEib/KNXEibSec protocol

Como podemos ver en la figura 40, el módulo tiene dos puertas tanto de entrada como de salida.

El módulo está conectado al modulo de aplicación, por donde recibe y envía los mensajes de tipo aplicación. Cuando recibe el mensaje de aplicación, en primer lugar verifica si la dirección de grupo que ha recibido está contenida en su tabla de direcciones de grupo a la que pertenece. Una vez comprobado que la dirección de grupo está contenida en la tabla, extrae la orden a enviar y junto con su dirección individual, que es la correspondiente al dispositivo, forma la trama que tiene que enviar a la red.

Aquí existe una diferencia dependiente del dispositivo que se utilice (según la red simulada). Si es la red EIB sin seguridad se compone la trama EIB cuyo formato es binario, se divide en caracteres para almacenarse en el mensaje *KNXEib_frame* y posteriormente enviarse al bus de línea. Mientras que si la red es EIBSec, se compone la trama entera de la misma forma pero sin dividir en caracteres. Se coge la parte de la trama que forma la unidad de la capa de transporte (T-PDU) junto con el campo de check. Con esta parte de datos se calcula un código de redundancia cíclica de 32 bits que se añade a la parte de los datos. Esto junto con el contador de 128 bits recibido del ACU, se hace XOR. El resultado de la operación XOR se cifra mediante cifrado AES128 con la clave propia del dispositivo. El resultado de la operación de cifrado es lo que se incluye en la parte de datos del mensaje *KNXEibSec_message.* Los demás parámetros del mensaje son completados a partir de las cabeceras de las tramas LDPU y NPDU que se han excluido del cifrado. La trama resultante

se envía a través de la puerta de salida conectada hacia el bus. El bus será el encargado de reenviarlo por la red hasta que llegue al dispositivo destino.

Uno de los mensajes que llega al módulo de protocolo por la conexión entre éste y el módulo del bus, cuyo funcionamiento se detalla más adelante, son los mensajes de estado del Bus llamados BUSY. Estos mensajes son recibidos cuando el modulo de protocolo envía un mensaje y el bus está ocupado. Los mensajes BUSY son recibidos como indicador de que el Bus está ocupado transmitiendo. En este caso se debe hacer una retransmisión del mensaje enviado anteriormente a la recepción de BUSY. La recepción del mensaje de estado del Bus y el comportamiento a realizar cuando el mensaje es recibido es exactamente igual para ambos modos de simulación.

En el caso del módulo de protocolo para la red no segura, *KNXEib_protocol,* existe otro tipo de mensaje que le llega por la puerta conectada entre el este y el bus. Este mensaje es el de la trama EIB o *KNXEib_frame.* Cuando este mensaje es recibido en el modulo de protocolo, en primer lugar comprueba el tipo de dirección destino del mensaje. Si la dirección destino extraída del mensaje es de grupo se realiza una búsqueda en la tabla de direcciones de grupo. En el caso de que sea una dirección individual directamente la comprueba con su dirección que es la del dispositivo. Una vez hechas las comprobaciones, si el resultado es positivo se extraen los caracteres de la trama y se compone la LDPU completa. De la trama binaria, se cogen los ultimo ocho caracteres correspondiente al campo check de la LDPU para realizar la comprobación de los datos recibidos. Para la comprobación de los datos, se calcula de nuevo el campo check y se compara con el que se ha recibido en la trama. Si los datos son correctos, extrae la parte de los datos de la trama EIB que tiene que ejecutar el módulo de aplicación y compone el mensaje de aplicación. Este mensaje será enviado por la conexión entre el modulo de protocolo y el de aplicación.

Como se ha comentado anteriormente, en el caso de una red segura se hace uso del modulo de protocolo *KNXEibSec_protocol.* En este modulo el mensaje recibido en la conexión con el bus es el mensaje en modo contador de la red EIBSec. Este mensaje es el llamado *KNXEibSec_message.* Una vez recibido este mensaje el procedimiento de comprobación de la dirección destino es el mismo. Una vez comprobado que el mensaje va destinado al dispositivo que contiene este modulo de protocolo, se extrae del mensaje el campo de datos. Primeramente, hay que realizar un descifrado de los datos ya que al tratarse de una red segura los mensajes vienen cifrados. Una vez realizado el proceso de descifrado AES128 y XOR de los datos descifrados junto con el

contador, se comprueba el CRC32. Si el CRC32 es correcto, se compone la LPDU con todos los campos. Una vez con la LDPU construida se realizan las mismas comprobaciones y se realiza el mismo procedimiento que en el modo no seguro.

5.2.4 Dispositivo KNX/EIB – KNXEib_device

KNXEib_device es semejante al dispositivo EIB y a su comportamiento real. Es un módulo compuesto de OMNeT++. Está formado por dos módulos simples: aplicación y protocolo.

En su conjunto este módulo intercambia mensajes de salida y entrada con el bus, y de entrada con el usuario. A nivel de red y de usuario, quien intercambia los mensajes con el bus y con quien actúa el usuario es el dispositivo EIB, al igual que sucede en la vida real.

Como se ha comentado en la parte de diseño de la simulación, igual que el módulo protocolo, aquí tendremos dos tipos de dispositivos: seguro y no seguro. El dispositivo no seguro es el que se utiliza para la simulación de la red EIB. Se considera no seguro porque el submódulo protocolo incluido en el dispositivo es *KNXEIB*_protocol. Al igual sucede en el caso del dispositivo seguro que está compuesto por el modulo protocolo con seguridad habilitada. Por lo tanto, un dispositivo seguro es que realizara el cifrado y descifrado de los mensajes para el envío de mensajes seguro a través de la red.

Lo único que diferencia a los dos tipos de dispositivos es el nombre del módulo y el nombre del submódulo protocolo. A continuación se muestra uno de ellos, dispositivo no seguro, como ejemplo de definición de módulo en el fichero NED:

```
// Modulo compuesto KNXEib_device
// Simula un dispositivo que puede ser un actuador o un sensor. Este elemento
compuesto está formado por: la parte que simula la implementación del
// Protocolo EIB y la parte de aplicación que es la que recibe las ordenes del
usuario.
// Parameters:
// string individualAddress: Dirección individual del elemento.
// string groupAddres: Dirección de grupo del elemento.
// string applicationType: tipo de aplicación EIB.
// bool actuator: indica si es un actuador que recibe las ordenes o un
interruptor que las ejecuta.

module KNXEibSec_device
{
    parameters:
        string individualAddress;
        string groupAddress;
        string individualAddressActuators;
        string applicationType;
        bool actuator;
```

```
        @display("b=90,170;i=misc/switchOff");
    gates:
        input in_bus;
        input in_user;
        output out_bus;
    submodules:
        protocol: KNXEibSec_protocol {
            parameters:
                individualAddress = individualAddress;
                groupAddress = groupAddress;
                applicationType = applicationType;
                @display("p=55,125;i=block/layer,cyan");
        }

        application: KNXEib_application {
            parameters:
                actuator = actuator;
                applicationType = applicationType;
                groupAddress = groupAddress;
                @display("p=55,55;i=block/process,grey");
        }

    connections:
        //conexiones externas
        out_bus <-- { @display("m=s"); } <-- protocol.out_bus;
        in_bus --> { @display("m=s"); } --> protocol.in_bus;
        in_user --> application.in_user;
        //conexiones internas
        protocol.out_application --> { @display("m=n"); } -->
application.in_application;
        protocol.in_application <-- { @display("m=n"); } <--
application.out_application;

}// end KNXEib_device
```

Como podemos ver en la definición del módulo, tenemos dos tipos de conexiones: las externas que son las que a priori ve el usuario (conexiones del módulo de simulación de usuario y las del módulo bus), y las internas que son las que no se ven (conexiones entre el módulo de aplicación y el módulo de protocolo).

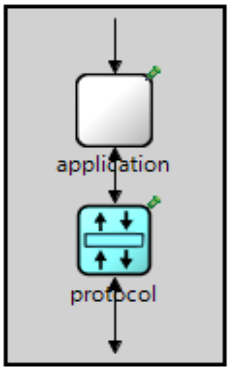


Figura 41 KNXEib/KNXEibSec device

En la simulación de la red el dispositivo tiene una representación, que dependerá del estado y del tipo de aplicación, que se puede ver en las Tablas 8

y 9 de este documento (véase Tabla 8 Símbolos interruptores y Tabla 9 Símbolos Actuadores). Si se quiere ver el detalle de este módulo se pincha sobre el dispositivo deseado y nos aparece una imagen igual en todos los dispositivos, que es la imagen detallada del módulo que aparece en la Figura 41 KNXEib/KNXEibSec device.

5.2.5 Bus KNX/EIB – KNXEib_bus

Es un módulo simple definido de la siguiente manera:

```
// Modulo Simple KNXEib_bus
// Es implementado como un hub que reenvía los mensajes que le llegan
por las demás puertas que tiene conectadas.
// Tendrá tantas conexiones, de entrada y salida, como elementos tenga
conectados
// Parameters:
// txRate: velocidad de transmisión de la línea

simple KNXEib_bus
{
    parameters:
        string txRate;
        @display("i=device/lan-bus_l");
    gates:
        input in[];
        output out[];

}//end KNXEib_bus
```

El único parámetro que hay que definir en la red para este módulo es *txRate* que es la velocidad de transferencia del bus en bits por segundo (bps).

Tendrá dos tipos de conexiones: las de entrada y las de salida. A efectos del bus todos los dispositivos conectados son iguales ya que únicamente se limita al reenvío de los mensajes que le llegan. Tiene tantas conexiones como dispositivos tenga la red mas el acoplador de línea. El acoplador de línea, que veremos a continuación, es considerado como otro dispositivo más. Aunque el módulo del acoplador realice funciones diferentes a las de los dispositivos para el bus es completamente transparente. La finalidad del bus es conectar todos los dispositivos de las líneas sin diferenciación alguna.

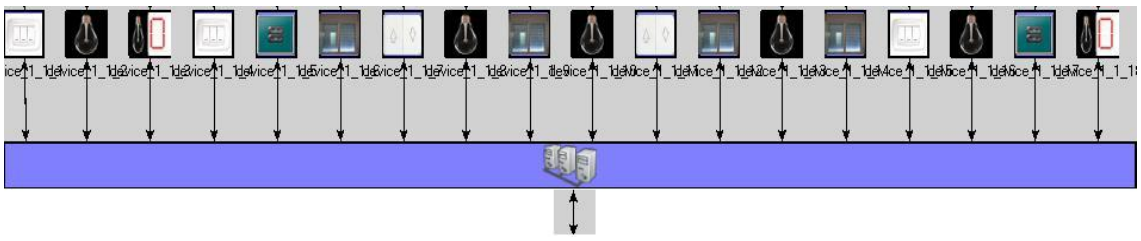


Figura 42 KNXEib_bus

El bus diferencia entre varios tipos de mensajes que le llegan a través de las puertas o conexiones que son: KNXEib_frame, KNXEibSec_message o securityMessage. En el módulo de Bus no se ha creado diferente para cada caso de simulación porque su funcionamiento es idéntico independientemente del tipo de mensaje que le llegue. Su finalidad será reenviar todos los mensajes que le lleguen por todas sus conexiones menos por la que le ha llegado el mensaje. Este comportamiento, es debido al funcionamiento real de la línea EIB. El funcionamiento del bus de línea en una red real es el reenvío del mensaje que le llega a todos los dispositivos que tiene conectados menos al que le envía el mensaje.

Hay un evento adicional creado por el bus, mensaje de estado, que se envía únicamente en el caso de que el bus este ocupado transmitiendo otro mensaje que le haya llegado anteriormente. En este caso transmite un mensaje BUSY únicamente al dispositivo que le ha mandado la trama cuando está ocupado transmitiendo.

Otra función importante en el bus que se ha implementado aparte de reenviar los mensajes, es la calcular y recolectar ciertas medidas estadísticas que grabaran para su posterior análisis. Estas estadísticas, medirán la carga y ocupación de la línea.

Los cálculos se realizan en función del estado del bus: ocupado o libre y la cantidad de paquetes en bits que transmite. Cada vez que le llega un mensaje, va calculando los datos de colisiones (ocupado) y de carga (libre), y cuando haya pasado un intervalo de tiempo establecido (1 segundo) hace la escritura de los datos a los correspondientes vectores estadísticos.

La carga se calculada por la función desarrollada dentro del módulo y es:

addStatistics (simtime_t nowTime, double timePacket, size_t collision)

Los parámetros de esta función son: *nowTime* es el instante de la simulación en el ha llegado un mensaje al bus, *timePacket* es el tiempo de envío del paquete, *collision* es el número de colisiones que se han producido.

Por ejemplo, cuando llega un mensaje al bus y está libre se invoca a la función de la siguiente forma *addStatistics(nowTime, busyTime, 0),* donde *nowTime* es el instante de tiempo que se ha calculado cuando ha llegado el mensaje en la función *handleMessage*, *busyTime* es el tiempo que se ha calculado que tarda el paquete en ser enviado a través de la red y 0 significa que no se han producido colisiones. Por otro lado, cuando llega el mensaje y el

bus está ocupado se hace una llamada del tipo *addStatistics(nowTime, 0, 1)* donde el tiempo del paquete es cero ya que no se puede enviar debido a que está ocupado y el número de colisiones es uno porque el estado del bus es ocupado. Este caso se maneja como si hubiese una colisión en la línea, como se describe en el funcionamiento del protocolo EIB. Esto mismo se detalla en el fragmento de código implementado en el módulo:

```
simtime_t nowTime = simTime();
//BUS OCUPADO
if (nowTime < intervalBusyTime)
{
	counterBusy ++;

	for (int i = 0; i < this->gateSize("in"); i++)
	{
		if (this->findGate("in", i) == rxGateIndex)
		{
			cMessage *msgBusy;
			msgBusy = new cMessage("BUSY");
			send (msgBusy, "out", i);
			addStatistics(nowTime, 0, 1);
			bubble ("BUSY");
			ev << "Trama descartada. Bus ocupado." << endl;
			delete msg;
		}
	}

}
//BUS LIBRE
else
{
	double busyTime = calculateBusyTime (lengthBytesSend);

	//calcula intervalo de tiempo que el bus estara ocupado tx según la
	longitud de la trama
	addStatistics(nowTime, busyTime, 0);
	intervalBusyTime = nowTime + busyTime;
	sendMessages(rxGateIndex, delayTime, msg, typeMsg);
	ev << "Mensaje enviado a todos los dispositivos conectados al Bus" <<
endl;

		}
```

Dentro de la llamada a la función que añade las estadísticas, se almacena el último instante de tiempo que los datos fueron grabados. De este modo, si la llamada a la función se ha producido antes del intervalo de tiempo de un segundo que se ha estipulado, únicamente se suman los nuevos datos que se han pasado como parámetro a la función. En cambio si el intervalo de tiempo estipulado ha pasado, después de incrementar los nuevos datos a los ya existentes, se invoca a la funciona que escribe las estadísticas en los vectores. Se graba el instante de tiempo y los datos calculado. Una vez grabado los datos se resetean los valores a cero para seguir haciendo los cálculos para el siguiente intervalo de tiempo.

Un fragmento de la función que añade las estadísticas explicadas anteriormente es:

```
//No ha pasado ningún intervalo de tiempo de muestra
	if (nowTime <= (lastInterval + monitoringInterval))
	{
		lastBusyTime += timePacket;
		lastColisions += collision;
	}
	//Ha pasado un intervalo de tiempo de muestra
	else
	{
		load = lastBusyTime / monitoringInterval;
		lastColisions += collision;
		writeStatistics(lastInterval,load, lastColisions);
		lastBusyTime = timePacket;
		lastInterval += monitoringInterval;

	}
```

Otro punto importante a destacar de este componente, es el reenvío de los mensajes a todos los dispositivos que tiene conectados. El reenvío se hace en base a un tiempo de retardo, que es calculado en función de la longitud en bytes de la trama y de la velocidad de transmisión que ha sido especificada como parámetro en el bus (generalmente 9600 que es lo estipulado en el estándar KNX/EIB).

5.2.6 Línea KNX/EIB – KNXEIB_line

Este módulo es un modulo compuesto en OMNeT++. Como se ha explicado en la introducción de este capítulo, hay dos tipos de líneas cuya base es idéntica para ambas. El módulo base de una línea como mínimo está formado por un submódulo que es el bus. Opcionalmente, la línea puede estar compuesta por dispositivos conectados al bus.

De acuerdo con los submódulos que componen la línea podemos distinguir dos tipos de línea: línea principal o *main line* y líneas que contienen los dispositivos de la red. La línea principal se compone únicamente de un submódulo que es el *KNXEib_bus.* Por otro lado están las líneas 1.1 y 1.2, simuladas en el presente proyecto, que está compuesta aparte del bus por todos los dispositivos que se han creado en la simulación de la red.

La definición de la línea principal desarrollada para la simulación del proyecto en el fichero NED es la siguiente:

```
// Modulo compuesto KNXEib_line_1_0
// Simula la línea EIB junto con todos los dispositivos que penden de ella.
Esta línea será una línea principal (Main line) de un área.
// Como es la línea principal solo estara compuesta por el bus.
// Parameters
// numeric const numCouplers: número de acopladores conectados a la línea.
```

```
// Submodules:
// KNXEib_bus: simulate bus de línea.
//
module KNXEib_line_1_0
{
    parameters:
        int numCouplers;
        @display("b=1000,50;m=w;bgb=1032,81");
    gates:
        input in_line[];
        output out_line[];
    submodules:
        bus_1_0: KNXEib_bus {
            parameters:
                txRate = "9600";
                @display("p=515,40;b=952,39");
        }
    connections:
        for i=0..numCouplers-1 {
            out_line[i] <-- bus_1_0.out++;
            in_line[i] --> bus_1_0.in++;
        }

}// end KNXEIB_line_1_0
```

Un fragmento de la definición de una de las líneas secundarias implementada en la simulación es:

```
//Los módulos que a continuación se definen son los diferentes elementos que
van a componer la red

// Módulo compuesto KNXEIB_line_1_1
// Simula la línea EIB junto con todos los dispositivos que penden de ella
// Parameters
// numeric const numCouplers: número de acopladores conectados a la línea.
//
module KNXEib_line_1_1
{
    parameters:
        int numCouplers;
        @display("b=500,50;m=w;bgb=1350,337");
    gates:
        input in_line[];
        output out_line[];
    submodules:
        bus_1_1: KNXEib_bus {
            parameters:
                txRate = "9600";
                @display("p=665,256;b=1256,50");
        }

      device_1_1_1: KNXEibSec_device {
            parameters:
                individualAddress = "1.1.1";
                groupAddress = "0/0/0=switchingAll";
                individualAddressActuators = "";
                applicationType = "switch1";
                actuator = false;
                @display("p=60,120;b=47,50");
        }
        device_1_1_2: KNXEibSec_device {
            parameters:
                individualAddress = "1.1.2";
                groupAddress = "1/1/1=light;1/1/7=light";
                individualAddressActuators = "";
                applicationType = "light1";
```

```
                actuator = true;
                @display("p=130,120;b=47,50");
        }

        device_1_1_3: KNXEibSec_device {
            parameters:
                individualAddress = "1.1.3";
                groupAddress = "1/1/10=dimming;1/1/11=dimming";
                individualAddressActuators = "";
                applicationType = "dimmingLight1";
                actuator = true;
                @display("p=200,120;b=47,50");
        }       user: userSimulator {
           parameters:
               @display("p=665,22;b=1256,42");
        }
    connections:
        device_1_1_1.out_bus --> bus_1_1.in++;
        device_1_1_1.in_bus <-- bus_1_1.out++;
        device_1_1_2.out_bus --> bus_1_1.in++;
        device_1_1_2.in_bus <-- bus_1_1.out++;
        device_1_1_3.out_bus --> bus_1_1.in++;
        device_1_1_3.in_bus <-- bus_1_1.out++;

        //Conexión del Line Coupler
        out_line++ <-- bus_1_1.out++;
        in_line++ --> bus_1_1.in++;

        //Conectamos el usuario a cada actuador
        device_1_1_1.in_user <-- user.out++;
        device_1_1_2.in_user <-- user.out++;
        device_1_1_3.in_user <-- user.out++;
}// end KNXEIB_line_1_1
```

Como podemos observar en los fragmentos de códigos de definición de módulo anteriores, tanto en la línea principal como en la secundaria, tan solo tiene un parámetro que es necesario indicar al definir una línea. Este parámetro, llamado *numCouplers,* es el número de acopladores conectados a la línea. En la línea principal se definen tanto acopladores como líneas secundarias (con dispositivos) se hayan creado en la red. Para el caso de la línea secundaria, este parámetro es uno ya que por especificaciones del estándar, las líneas secundarias únicamente pueden estar conectadas a un acoplador.

Durante la definición del módulo que simula la línea, al igual que en todos los módulos compuestos, es necesario definir todos los parámetros de sus submódulos y hacer sus conexiones. En nuestro caso se hacen las conexiones que conectan el usuario con los dispositivos, así como la conexión entre el bus y los dispositivos (dispositivos y acoplador).

La representación grafica de una línea secundaria es la siguiente:

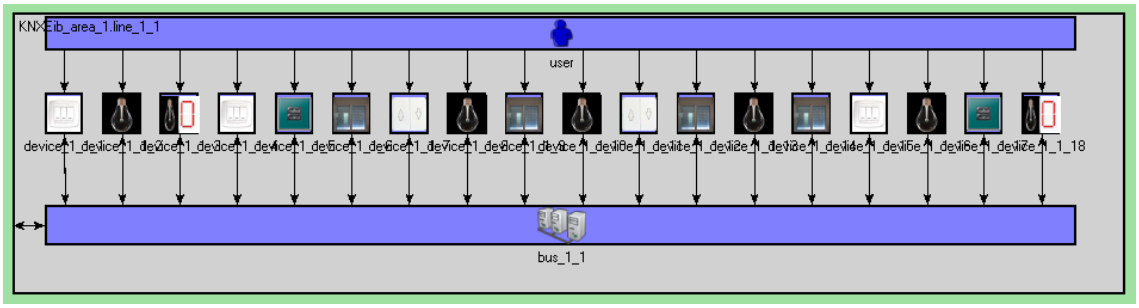


Figura 43KNXEib_line

Sin embargo, la representación de la línea principal es gráficamente más sencilla ya que únicamente dispone de la conexión del bus:

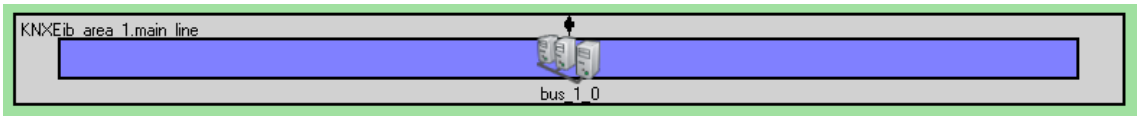


Figura 44 KNXEib Main Line

5.2.7 Acoplador de línea KNX/EIB – KNXEib_deviceCoupler y KNXEibSec_deviceACU

La funcionalidad de este módulo es similar a la del acoplador KNX EIB real, cuya finalidad es la de filtrar las PDUs dejando pasar únicamente las destinadas a su línea. Contiene una tabla de routing, donde se realiza una búsqueda para el caso de direcciones de grupo. Si se trata de una dirección individual, la compara con la suya dejando pasar el mensaje o descartando en otro caso. Con este método se consigue evitar la sobrecarga de la red y con ello evitar que se descarten tramas de dispositivos por colisión.

Como ya se comentó en el capítulo 3 de la presente memoria, el acoplador de línea es otro de los componentes de la red cuyo comportamiento varía en función del protocolo, EIB o EIBSec. Para este módulo en el caso no seguro, EIB, tenemos un acoplador de línea. Mientras que para el modo seguro, EIBSec, es un ACU. Las funciones del ACU son idénticas a las del acoplador de línea con un añadido de seguridad. Estos añadidos de seguridad son el manejo de los mensajes que informan a los dispositivos de la clave y el contador de grupo. Los nombres utilizados para ambos dispositivos son *KNXEibSec_deviceCoupler y KNXEibSec_deviceACU* respectivamente.

La implementación del acoplador a nivel de red dentro del fichero NED es idéntica para ambos tipos. A continuación se muestra el fragmento de código que define el componente acoplador:

```
// Módulo simple KNXEib_deviceCoupler
// Simula un acoplador que une dos líneas o bus, cuyo fin es filtrar las
tramas de esa línea
// Parameters:
//     string individualAddress: dirección individual del acoplador
//     string routingTable: tabla de enrutado o filtrado
//

simple KNXEib_deviceCoupler
{
    parameters:
        string individualAddress;
        string routingTable;
        @display("i=device/acoplador");
    gates:
        input in_line;
        input in_backbone;
        output out_line;
        output out_backbone;

}// end KNXEib_deviceCoupler
```

Figura 45 KNXEib_deviceCoupler / KNXEibSec_deviceACU

Los parámetros a especificar en este dispositivo son:

- *individualAddress:* dirección individual del dispositivo.
- *routingTable:* tabla que contiene las direcciones de grupo de su línea, utilizadas para el filtrado de los mensajes.

Como hemos dicho anteriormente la definición del modulo dentro del fichero donde se crea la red es exactamente igual. La diferencia entre ambos es en la funcionalidad desarrollada en el comportamiento del módulo, ya que si no existiera diferencia se podría utilizar la misma en ambos casos.

El dispositivo *KNXEibSec_deviceACU*, hará las mismas funciones que el acoplador de línea de una red EIB con un añadido más: maneja mensajes de seguridad que se intercambian entre este dispositivo y todos los dispositivos conectados a la línea. El ACU enviará dos tipos de mensajes de seguridad a cada dispositivo conectado al bus, *Join Group Response* que contiene la clave para cifrar el mensaje y *Group Resync Response* que contiene el contador para hacer XOR con los datos en modo contador.

El funcionamiento de este dispositivo es la siguiente: cuando llega un mensaje procedente de la línea verifica la dirección. Si es una dirección de grupo envía el mensaje al backbone directamente para que llegue al resto de

acopladores. Si es individual, la compara con la suya propia y en caso de coincidencia la descarta ya que el mensaje procede de la línea.

Cuando llega un mensaje procedente del backbone realiza la operación inversa: verifica si es individual para compararla con la suya y si coinciden la deja pasar sino la descarta ya que no va dirigido a su línea. Si es de grupo la compara con las direcciones de grupo de su tabla. Dependiendo del resultado de la comparación deja pasar el mensaje hacia la línea en el caso de que la dirección de grupo este contenido. En caso contrario la trama es descartada.

5.2.8 Red KNX/EIB – KNXEib_network y KNXEibSec_securityNetwork

Es la red que simulamos. En el caso de nuestro proyecto hay dos redes: red KNXEib_network y red KNXEibSec_securityNetwork. La primera red es la EIB sin seguridad habilitada mientras que la segunda es la red EIBSec con seguridad.

La declaración de red es un modulo compuesto de OMNet++. Está compuesto por todos y cada uno de los módulos analizados hasta ahora en la memoria. Se diferencia entre modo no seguro y seguro por su comportamiento que difiere en dos componentes básicos. Estos componentes básicos son el modulo de protocolo y acoplador. A continuación vemos un ejemplo de definición de red, modo seguro, extraído del proyecto:

```
// Modulo compuesto KNXEib_network
// Simula un área EIB, que está compuesta por una línea principal, acopladores
de línea (line coupler) y líneas.
// Parameters
//           - numeric const numCouplers: número de acopladores conectados a
la línea.
//

network KNXEib_network
{
    parameters:
        int numCouplers = 0;
        @display("bgb=1032,379");
    gates:
        input in_line[];
        output out_line[];
    submodules:
        main_line: KNXEib_line_1_0 {
            parameters:
                numCouplers = 2; // 2 LC (EN EL CASO DE QUE HAYA MAS AREAS
HABRA TANTOS LC COMO LINEAS TENGAMOS + 1BC PARA CONECTARLO AL BACKBONE)
                @display("p=511,60;b=966,50");
        }
        //devices
        line_1_1: KNXEib_line_1_1 {
            parameters:
                numCouplers = 1;
                @display("p=235,310;b=415,50");
```

```
        }
        line_1_2: KNXEib_line_1_2 {
            parameters:
                numCouplers = 1;
                @display("p=784,310;b=419,50");
        }
        line_coupler_1: KNXEib_deviceCoupler {
            parameters:
                individualAddress = "1.1.0";
                routingTable =
"0/0/0;1/1/1;1/1/7;1/1/5;1/1/3;1/1/2;1/1/4;1/1/6;1/1/9;1/1/10;1/1/11;1/1/12";
                @display("p=236,178");
        }
        line_coupler_2: KNXEib_deviceCoupler {
            parameters:
                individualAddress = "1.2.0";
                routingTable =
"0/0/0;1/1/7;1/1/5;1/1/2;1/1/4;1/1/6;1/1/8;1/1/10;1/1/12;1/1/13";
                @display("p=785,178");
        }
    connections:
        //Conexión de los acopladores de línea a la línea
        line_1_1.in_line++ <-- line_coupler_1.out_line;
        line_1_1.out_line++ --> line_coupler_1.in_line;
        line_1_2.in_line++ <-- line_coupler_2.out_line;
        line_1_2.out_line++ --> line_coupler_2.in_line;

        //Conexión de los acopladores de línea a la línea principal
        line_coupler_1.in_backbone <-- main_line.out_line++;
        line_coupler_1.out_backbone --> main_line.in_line++;
        line_coupler_2.in_backbone <-- main_line.out_line++;
        line_coupler_2.out_backbone --> main_line.in_line++;

        for i=0..numCouplers-1 {
            out_line[i] <-- main_line.out_line++;
            in_line[i] --> main_line.in_line++;
        }
}//end KNXEib_network
```

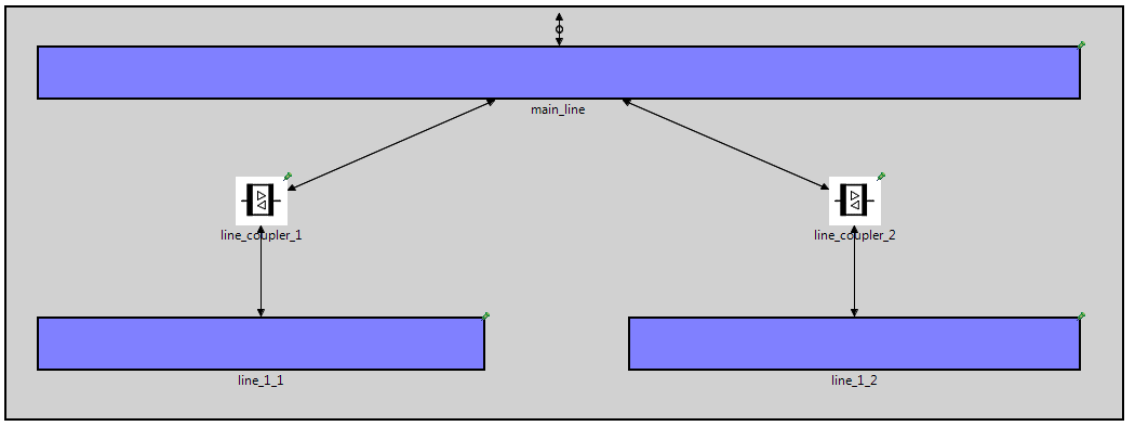


Figura 46 KNXEib_network

Como podemos apreciar en el código anterior, cuando se define la red final a simular se hacen las conexiones de los submódulos que lo componen. Se realiza la conexión entre la línea principal y el acoplador, así como entre los acopladores y sus líneas secundarias.

Un parámetro que se define en la declaración de la red es el número de acopladores. Estos acopladores, son los acopladores de backbone. El número de acopladores de backbone es el número total de líneas principales que haya simuladas.

Por simplicidad, en el proyecto se ha optado por simular una red consistente en un área. Está compuesto por una línea principal conectada a dos acopladores de línea. Cada acoplador consiste en una línea compuesta por el bus, los dispositivos y el usuario como se ha explicado anteriormente. La línea principal tiene tantos acopladores de línea como líneas hay, ya que cada acoplador de línea filtrará los mensajes procedentes de otras líneas hacia la suya.

Teniendo en cuenta que tenemos simulada un área, no existen ni las líneas de backbone ni los acopladores de backbone. Por lo tanto el parámetro que indica el número de acopladores es cero. Si se representan más áreas, tienen que definirse un número total de acopladores de backbone igual al número de áreas que se han definido. La función del acoplador de backbone es la misma que la del acoplador de línea: filtrar los mensajes había su área.

Cuando se definen los acopladores en la red se define la tabla de direcciones de grupo a los que pertenecen los dispositivos de su línea para hacer el filtrado de las tramas que llegan.

5.2.9 Terminador de Red – KNXEib_Terminator

Este modulo es común y general para ambas redes. Se ha implementado este modulo para el caso en que se quiera probar el funcionamiento de ciertas partes de la red sin tener que construir toda la red entera. Por ejemplo, en caso de que se quiera probar el funcionamiento del protocolo, requiere estar conectado al bus. Como realmente las pruebas son para verificar el funcionamiento de protocolo, se puede conectar el terminador de red en el lugar del bus. En este ejemplo el modulo *KNXEib_Terminator* simula la conexión del Bus pero no realiza su función. Este módulo sirve para hacer la terminación de la red, como su nombre indica, simulando la conexión pero descartando todos los mensajes que le llegan. Lo que se pretende con este modulo es satisfacer los requisitos de las conexiones definidas en cada dispositivo.

La definición de este dispositivo dentro de la red es:

```
// Modulo simple KNXEib_terminator
// Terminación de una línea cuando no utiliza un acoplador de línea
// Parameters:
```

```
//    No existen
//
simple KNXEib_terminator
{
    gates:
        input in_terminator;
        output out_terminator;

}// end KNXEib_terminator
```

Como podemos ver tiene dos conexiones definidas una de entrada y otra de salida para poder cumplir con los requisitos de conexiones. La conexión de entrada sirve para recibir los mensajes, mientras que por la conexión de salida no se envía nada. La funcionalidad de este módulo simple, es la de recibir mensajes y descartarlos.

5.3 OTRAS CLASES C++

Existe otra clase C++ de uso común a ambas redes. No es un módulo sino que se trata de una clase común con funciones complementarias de ayuda para la formación y chequeo de la tramas EIB en ambas simulaciones. Se llama *KNXEib_messageFunctions* y es usada principalmente por el módulo de protocolo aunque hay otros modulo que hacen pequeño uso de él.

A continuación se detallan los métodos de esta clase agrupados por funcionalidad:

- ***Métodos de generación de PDUs:***
  - *string makeLPDU (string repeat, string priority, string addressSource, string addressReceiver,bool groupAddress, string dataSend, string typeAPCI)* : genera la LPDU (trama nivel de enlace de datos) a enviar a través de la red EIB.
  - *string makeNPDU(string typeAPCI):* genera la NPDU (trama a nivel de red) encapsulada en la LPDU.
  - *string makeTPDU(string typeAPCI):* genera la TPDU (trama a nivel de transporte) encapsulada en la NPDU.
  - *string makeAPDU(string typeAPCI):* genera la APDU (trama a nivel de aplicacion) contenida en la TPDU.
- ***Métodos auxiliares de la trama EIB:***
  - *vector<string> makeCharacters (string LPDU):* prepara la trama de enlace de datos en caracteres para ser enviados a través de la red.

- *bool isGroupAddress(string address):* nos dice si la dirección que se le ha pasado como parámetro es de grupo o no.

- *bool checkTelegram (vector<string> telegram):* comprueba que el telegrama ha llegado correctamente, a través del check field.

- *string dataTypeRx (string data):* traduce los datos pasados como parámetros a tipo de datos, que es la orden que se ejecuta en la aplicación.

➢ ***Métodos auxiliares de conversión de datos:***

- *string convertAddress (bool groupAddress, string address):* convierte a caracteres es decimal la dirección que se le pasa como parámetro. Hace una tipo de conversión u otra dependiendo del parámetro indicador del tipo de dirección que se le ha pasado.

- *unsigned int binaryToDecimal (string binary):* convierte un número binario a decimal.

- *string decimalToBinary (string decimal, unsigned int length):* convierte un número decimal en binario con la longitud indicada en el parámetro.

➢ ***Métodos para extraer campos de la LDPU:***

- *string extractLPDU (vector<string> characters):* extrae de cada carácter que se ha recibido, la parte de los datos para componer la LPDU completa.

- *string extractData (string Lpdu):* extrae de la trama el campo de datos.

- *string extractAddressReceiver(vector<string> telegramRx):* extrae la dirección destino de los caracteres pasados como parámetro.

- *string extractAddressSource (vector<string> telegramRx):* extrae la dirección origen de los caracteres recibidos que forman la trama de datos.

➢ ***Métodos auxiliares de tramas EIBSec:***

- *string doCRC32(string data):* calcula el código de redundancia cíclica de 32 bits y lo adjunta a los datos que se le han pasado como parámetro.

- *uLong makeCRC32(unsigned char* buffer, int length):* calcula el CRC32 a partir de unos datos y una longitud pasados como parámetro.

- *bool checkCRC32(string buffer, int length, uLong crc):* chequea el campo CRC32 que se ha recibido junto con los datos. Verifica calculándolo de nuevo si es correcto o no comparándolo con el recibido.

- *string do_CounterConverter (string counter, size_t lengthCounter):* convierte el contador pasado como parámetro en un contador en binario de 128 bits.

- *string do_Xor(string counte, int longitudCounter, string dataToSend, int longitudDatos):* hace la XOR del contador y de los datos pasados como parámetro.

- *unsigned char *aes_encrypt(char *dataToSend, unsigned char *key, unsigned char *iv):* cifra en AES128 los datos pasados como parámetro, con la clave y el vector pasados como parámetro.

- *unsigned char *aes_decrypt(unsigned char *ciphertextRx, int length_msg, unsigned char *key, unsigned char *iv):* descifra los datos utilizando el vector y la clave pasadas como parámetros de los datos encriptados también pasados como parámetros a la función.

➢ ***Métodos para setear el valor de los campos de la LPDU***

- *void setRepetitionField (string repetition):* fija el campo de repetición de la LPDU.

- *void setPriorityTypeField (string priorityType):* fija el campo de tipo de prioridad de la LPDU.

- *void setControlField (string controlFieldFrame):* fija el campo de control de la LPDU.

- *void setAddressSourceField (string addressSource):* fija la dirección origen en la LPDU.

- *void setAddressReceiverField (string addressReceiver):* fija la dirección destino en la LPDU.

- *void setAddressReceiverFlagField(string addressReceiverFlag):* fija el tipo de dirección de destino en la LPDU.

- *void setRoutingCounterField (string routingCounter):* fija el contador de enrutado en la LPDU.
- *void setLengthField (string lengthFrame):* fija la longitud de la trama en el campo de LDPU.
- *void setTPCIField (string TPCI):* fija el campo TPCI en la LPDU.
- *void setAPCIField (string APCI):* fija el campo APCI en la LPDU.
- *void setDataField (string data):* fija los datos a enviar en la LPDU.
- *void setChecksumField (string checksum):* fija el campo de comprobación en la LPDU.

➢ ***Métodos para obtener los campos de la LPDU***

- *string getRepetitionField ():* obtiene el campo de repetición de la LPDU.
- *string getPriorityTypeField ():* obtiene el campo de tipo de prioridad de la LPDU.
- *string getControlField ():* obtiene el campo de control de la LPDU.
- *string getAddressSourceField ():* obtiene la dirección origen de la LPDU.
- *string getAddressReceiverField ():* obtiene la dirección destino de la LPDU.
- *string getAddressReceiverFlagField ():* obtiene el tipo de dirección de destino de la LPDU.
- *string getRoutingCounterField ():* obtiene el campo de contador de enrutado de la LPDU.
- *string getLengthField ():* obtiene la longitud de la trama de la LPDU.
- *string getTPCIField ():* obtiene el campo TPCI de la LPDU.
- *string getAPCIField ():* obtiene el campo APCI de la LPDU.
- *string getDataField ():* obtiene los datos a enviar de la LPDU.

- *string getChecksumField ():* obtiene el campo de comprobación de la LDPU.

- *int getLongitudMsg():* obtiene la longitud del mensaje de los datos encriptados.

Capítulo 6: DISEÑO Y DESARROLLO DE UN ESCENARIO DE SIMULACIÓN

Como ya se ha mencionado anteriormente, en la descripción de los módulos, se han desarrollado dos modelos de red: red KNX EIB (*KNXEIBSec_network)* y red KNX EIBSec (*KNXEibSec_securityNetwork)*. Esto nos permite simular ambos escenarios para realizar una comparativa entre ambas redes. La comparativa se realiza en función de la carga de la red, número de colisiones en la red, el número de paquetes que se envía, etc. En definitiva, el comportamiento de cada red simulada y las diferencias existentes entre ellas.

Las medidas han sido desarrolladas en el módulo del Bus (véase 5.2.5 Bus KNX/EIB – KNXEib_bus) que nos servirá para comparar la carga de la red en el caso de líneas seguras y líneas no seguras. Lo que queremos conseguir con estas simulaciones es la carga adicional cuando se utiliza el modo seguro. La causa del incremento de carga en la red segura es el aumento de mensajes enviados y el tiempo de procesado de la trama.

6.1 CONFIGURACIÓN SIMULACIÓN

El esquema de la red que va a ser simulada para ambos casos es la siguiente:

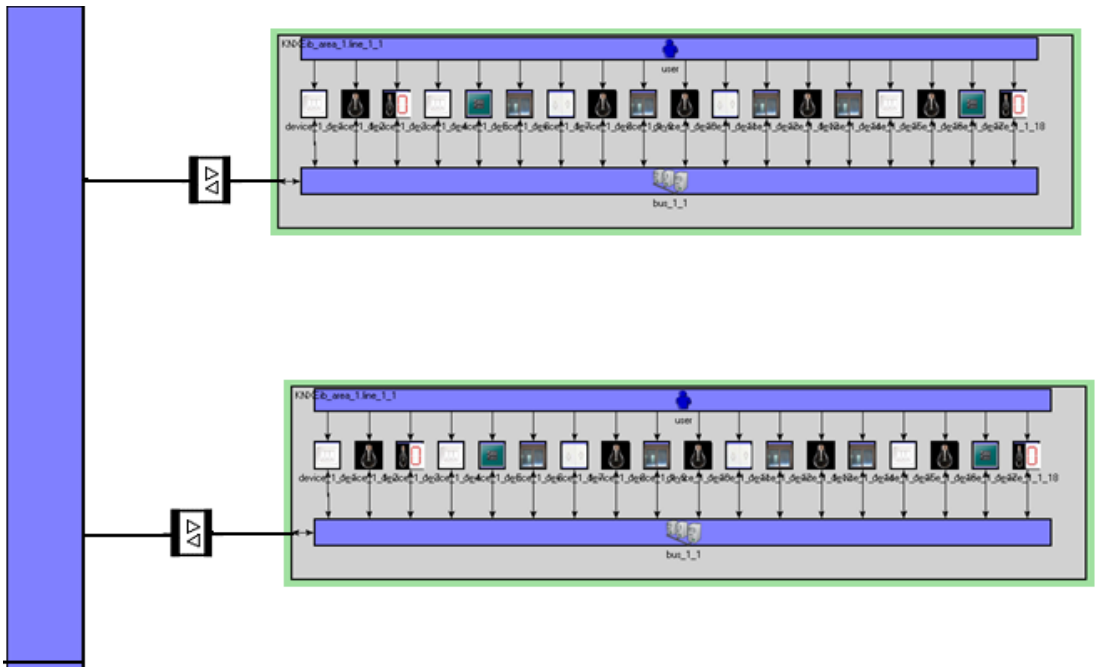


Figura 47 Esquema de modelo de Red simulado

La red consiste en 18 dispositivos por línea que contienen todos los dispositivos definidos en el proyecto: interruptor general, interruptor de luz, interruptor de persiana, regulador de luz, luz, persianas y luz regulada. Hay una sola línea principal, con lo que estamos representando lo que se llama en EIB un Área, de la cual penden dos líneas que contienen los 18 dispositivos antes referenciados. Entre la línea principal y las dos líneas secundarias tenemos dos acopladores de línea, uno por cada línea secundaria.

Cada interruptor tiene definidos: una tabla con las direcciones individuales y otra con las direcciones de grupo a los que está conectado. Cada dispositivo pertenece al menos a dos grupos.

Existe un grupo general, *switchingAll*, con el que se encienden todos los dispositivos que hayan sido configurados con ordenes ON/OFF como son las luces y las persianas. Las órdenes a este grupo solo pueden ser enviadas a través de un solo dispositivo en cada línea, que realiza funciones de interruptor general.

Existen otros tres grupos generales pero a nivel del tipo de actuador: uno para luz, otro para persianas y otro para luz regulada. Hay un sólo interruptor configurado en cada caso que puede ejecutar la orden a este grupo. En este caso cuando el interruptor envíe la orden al grupo, todos los actuadores correspondientes a ese tipo de interruptor ejecutaran las órdenes especificadas. Este grupo se corresponde con el que hemos llamado grupoA en cada tipo de actuador.

También hay otros grupos definidos de manera aleatoria que encienden, apagan o regulan a los actuadores.

Para simular la actividad de la red, el tráfico es definido a través del fichero de órdenes que lee el usuario (véase 5.2.1 Usuario simulado – userSimulator). Su configuración se hace en base a los tipos de grupos existentes y a los tipos de dispositivos que hay:

- configuración de tráfico general: encendido/apagado de persianas y luces activando los switch1 y switch4 de la red que son los que hacen de interruptor general.
- configuración de trafico general dependiente del tipo de dispositivo:
  - ✓ Encendido/Apagado de luces.
  - ✓ Apertura/Cierre de persianas.

- ✓ Subida/Bajada de intensidad de luz regulada.

- configuración de grupo aleatorio en cada caso.

El tiempo de simulación será dependiente del número de mensajes que tenga que enviar el usuario y el tiempo entre esos mensajes. Por ejemplo si tengo la siguiente línea de ejecución:

```
0  1.0   300   switch1     switchingAll      ON
```

se ejecutarían 300 mensajes cada segundo, con lo que el tiempo de ejecución sería de 300 segundos (5 minutos).

Las líneas medirán la carga que tiene su segmento a través de la funcionalidad implementada en el bus. Cada modo de simulación tendrá un fichero *.vec* que contendrá dos vectores de datos por cada línea, *<bus_name>-vectorLoad.vec* y *<bus_name>-vectorBusy.vec*, así como diferentes medidas escalares de la red. También al finalizar se graba un fichero .log de la red donde se puede leer y verificar todo el intercambio de mensajes y eventos ocurridos en la simulación.

6.2 PRUEBAS DEL SIMULADOR

Para comprobar y comparar el comportamiento de una red EIB, se han configurado 2 casos de pruebas diferentes.

- KNXEibSec_network: representa una red tradicional de EIB. Los dispositivos se comunican a través de una red no segura enviando mensajes sin cifrar. Solo un código de redundancia cíclica es incluida en la trama para su posterior verificación.

- KNXEibSec_securityNetwork: representa una red EIB con seguridad. Los dispositivos son seguros porque se ha implementado una funcionalidad adicional a EIB para el cifrado de los mensajes con una clave y contador. La clave y el contador son enviados previamente por el ACU que es un dispositivo implementado para redes EIBSec.

La duración de ambas será de 15 minutos de duración, con varios tipos de mensajes enviados. Las ordenes que realiza el usuario en ambas redes, configuradas en el fichero de lectura del usuario, son las siguientes:

```
#tiempo/intervalo_tiempo/contador, aplicacion, dirección, datos
0     3.0   300   switch1     switchingAll      ON
0     3.0   300   switch4     switchingAll      OFF
0     1.0   900   switch2     lightGroupA OFF
```

```
0     1.0   900   switch5     lightGroupB ON
0     2.0   450   switchBlind1      blindGroupA OFF
0     2.0   450   dimmer3     dimmingGroupA     UP
0     1.5   600   dimmer1     dimmingGroupA     DOWN
0     1.5   600   switch6     lightGroupE ON
0     2.0   450   dimmer4     dimmingGroupD     UP
0     2.0   450   dimmer2     dimmingGroupB     UP
0     3.0   300   switchBlind4      blindGroupC ON
0     3.0   300   switch3     lightGroupD ON
```

6.3 RESULTADOS DE LA SIMULACIÓN

Los resultados de la simulación se van a presentar con unos gráficos que muestran:

- *Carga de la red:* es la carga de la red calculada en tanto por ciento. El 100% de la carga no se podría alcanzar ya que el comportamiento del protocolo obliga a espera un tiempo para el reenvío. Esto supone que todos los dispositivos no pueden enviar a la vez, ya que el Bus está ocupad durante el tiempo de transmisión de la trama.

- *Ocupación del Bus:* esta respuesta es enviada por el Bus cuando el módulo protocolo o el acoplador intentan enviarle una trama y el Bus está ocupado enviando otro mensaje. El gráfico nos indica cuantas veces en intervalos de un segundo el bus está ocupado cuando un dispositivo intenta enviarle una trama de datos. Esto nos indica la cantidad de mensajes que sufren retardo debido a la carga del bus.

En base a estas medidas, se hace la comparación entre una red segura y una red no segura mostrando las diferencias que hay entre ambas.

6.3.1 Resultados de simulación para modo no seguro

Las características de configuración de esta red es una red KNX EIB clásica, es decir, sin ninguna configuración de seguridad añadida tal y como se define en el estándar EIB.

Los resultados de simulación para la línea principal o main line son:

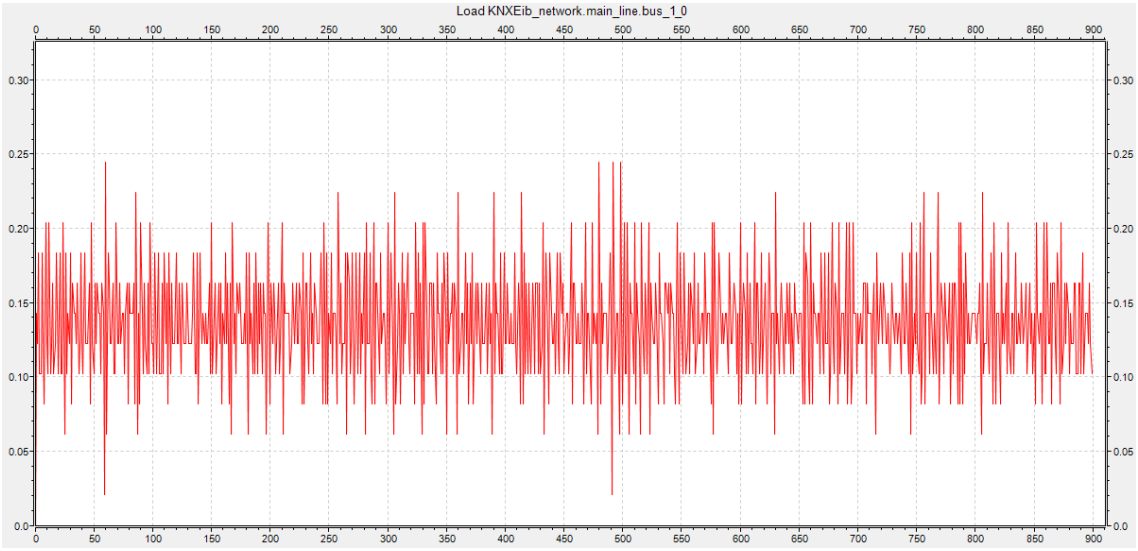


Figura 48 Carga de la línea principal main_line_1_0

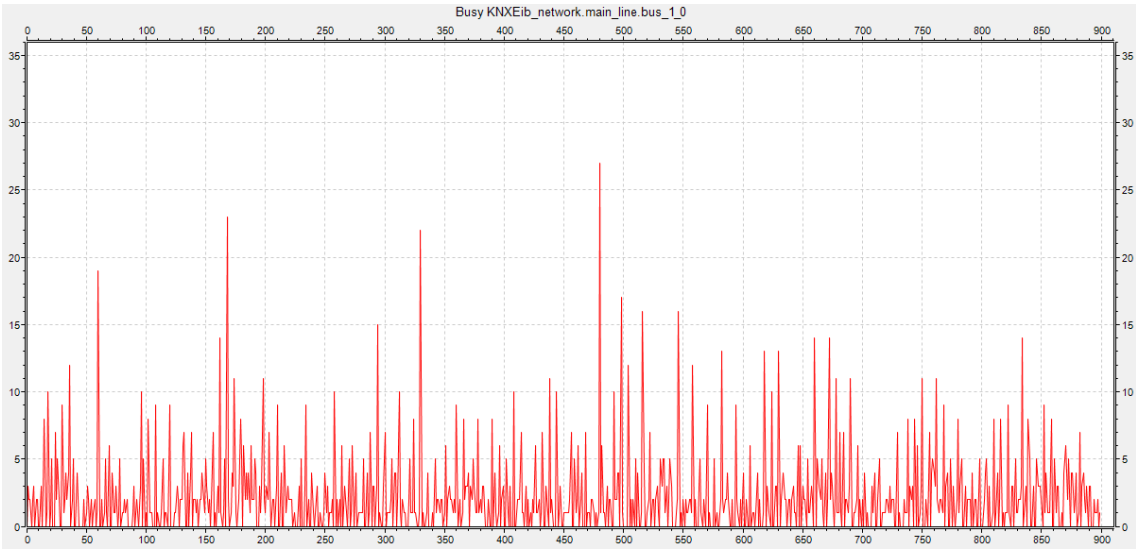


Figura 49 Respuestas Bus Ocupado main_line_1_0

Las gráficas resultantes de la línea 1.1:

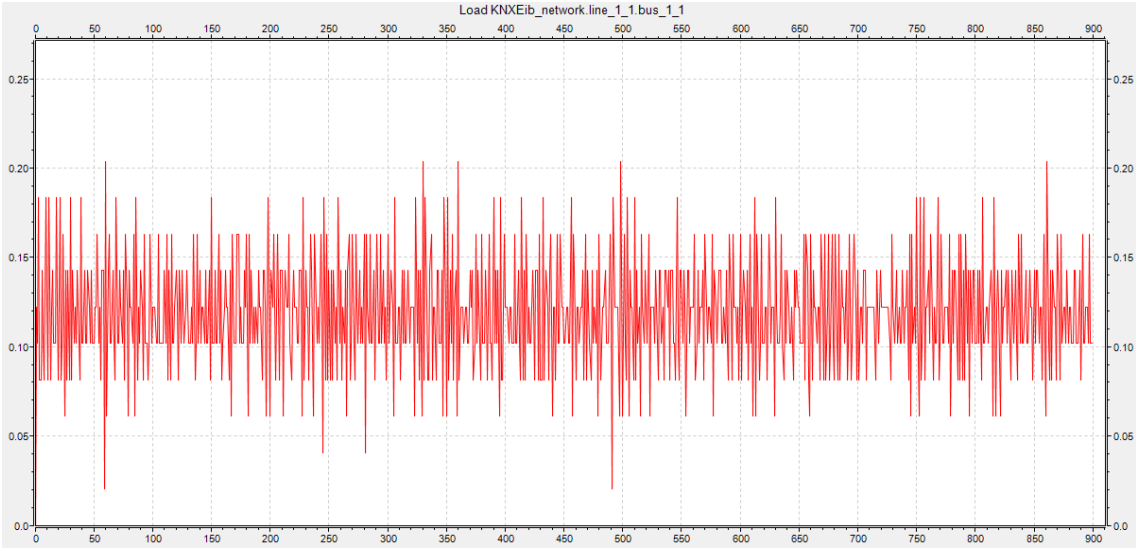


Figura 50 Carga de la línea line_1_1

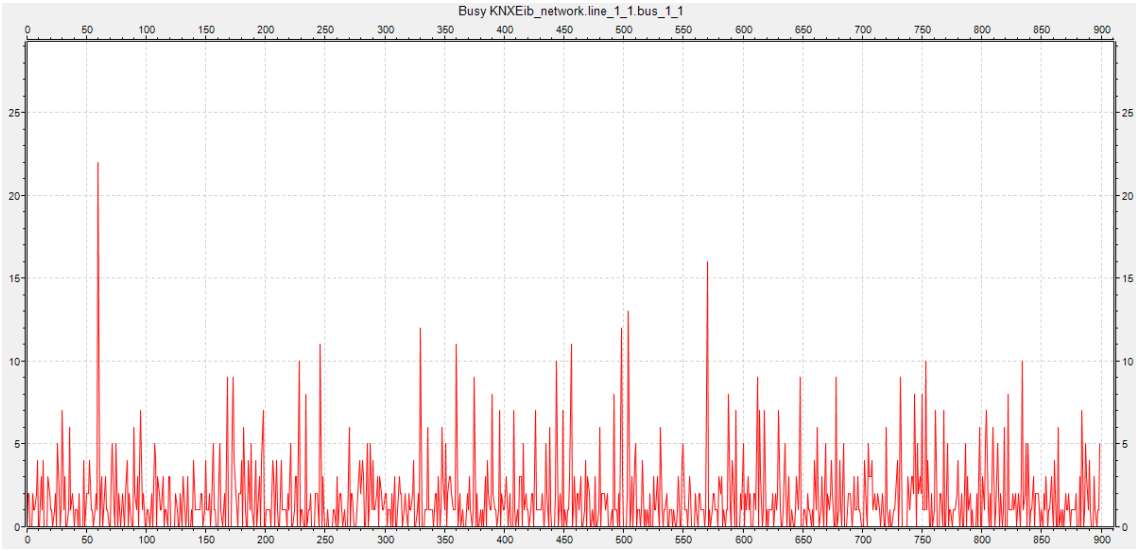


Figura 51 Respuestas Bus Ocupado de la línea line_1_1

Y por último, las gráficas resultantes de la línea 1.2 son:

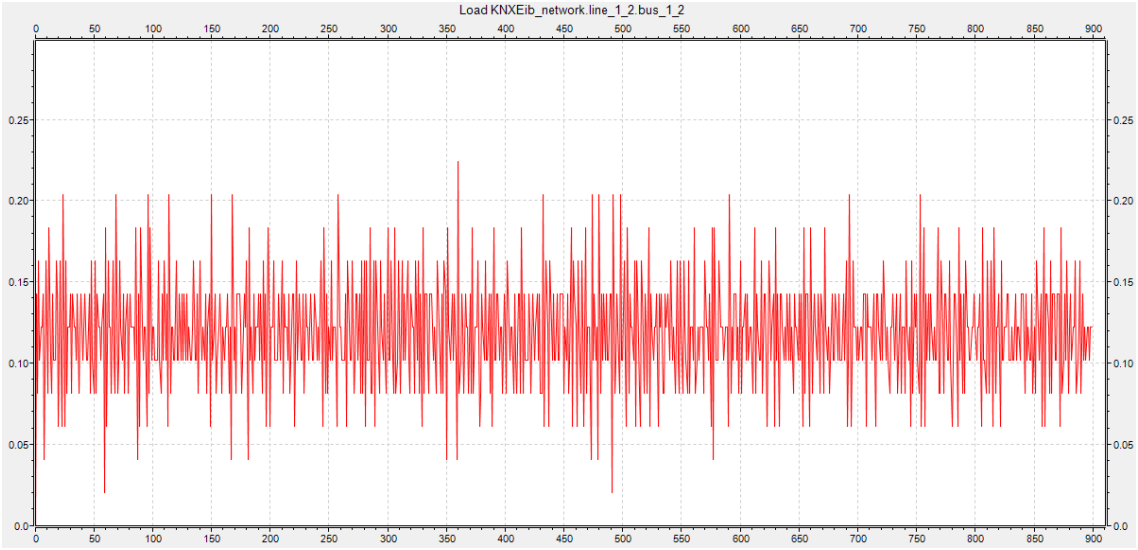


Figura 52 Carga de la línea line_1_2

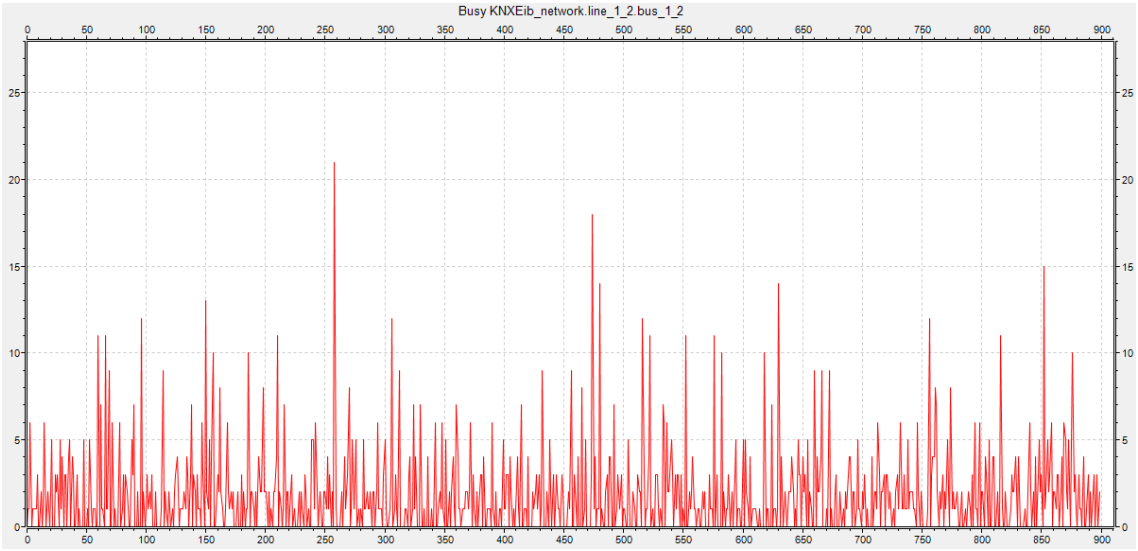


Figura 53 Respuestas Bus Ocupado de la línea line_1_2

6.3.2 Resultados de simulación para modo seguro

En esta sección se muestran los resultados para la misma configuración de red pero con la seguridad añadida al protocolo.

Los resultados de simulación para la línea principal o *main línea* son:

Figura 54 Carga de la línea principal main_line_1_0

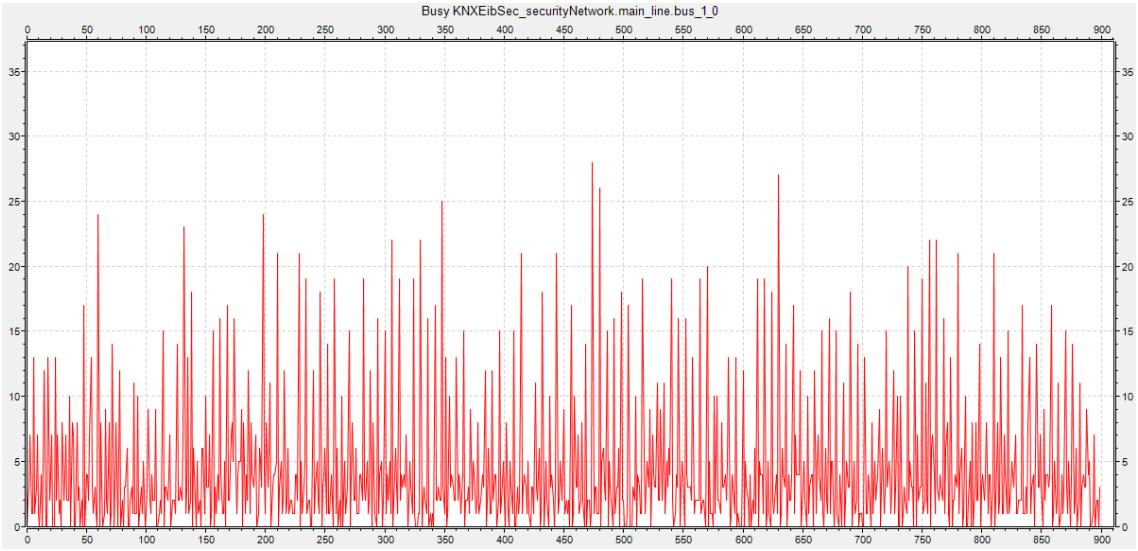


Figura 55 Respuestas Bus Ocupado main_line_1_0

Las gráficas para la línea 1.1:

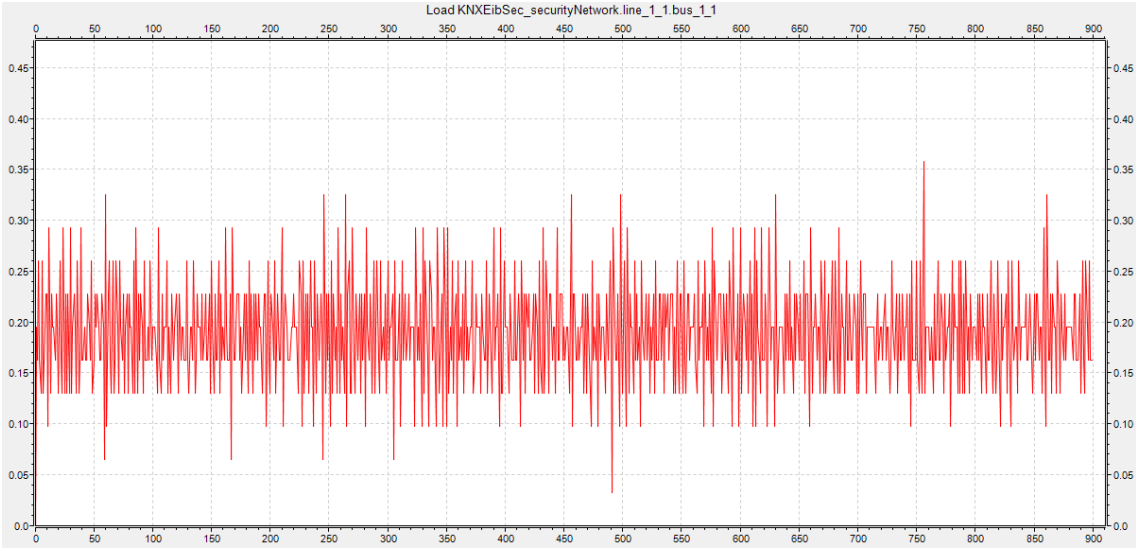


Figura 56 Carga de la línea line_1_1

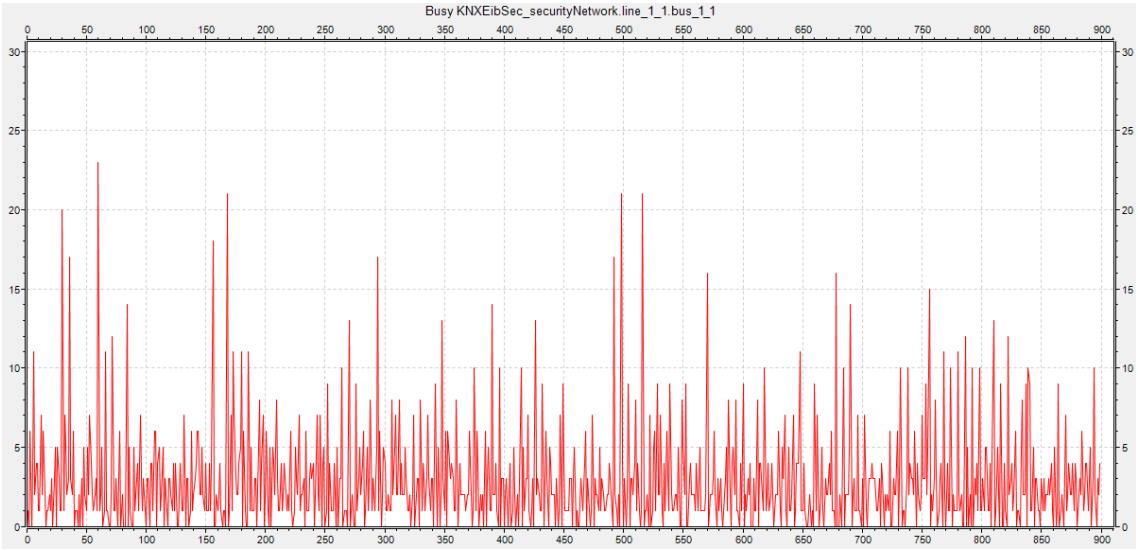


Figura 57 Respuestas Bus Ocupado línea line_1_1

Las gráficas para la línea 1.2:

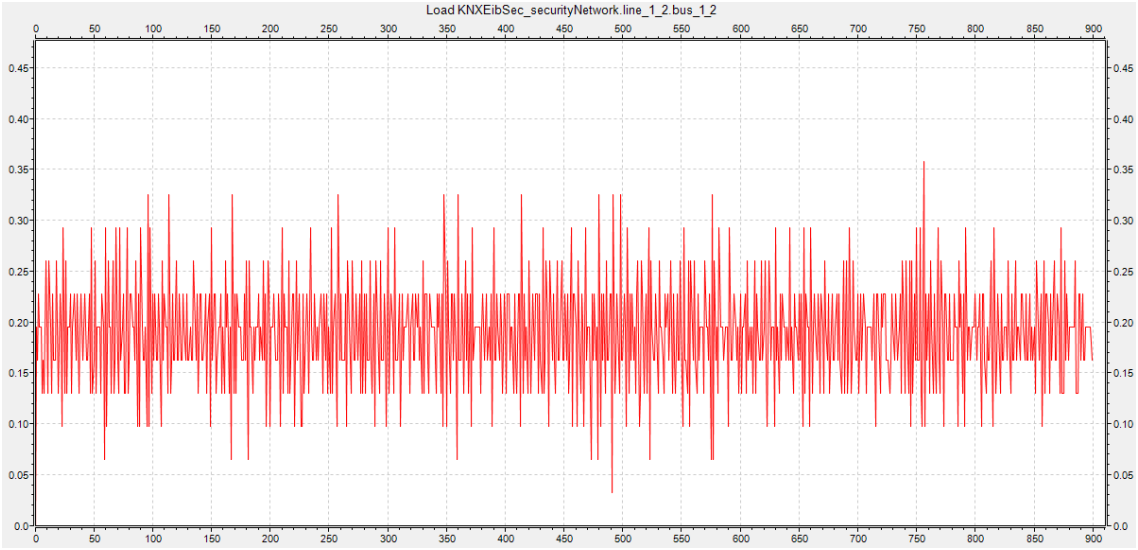


Figura 58 Carga de la línea line_1_2

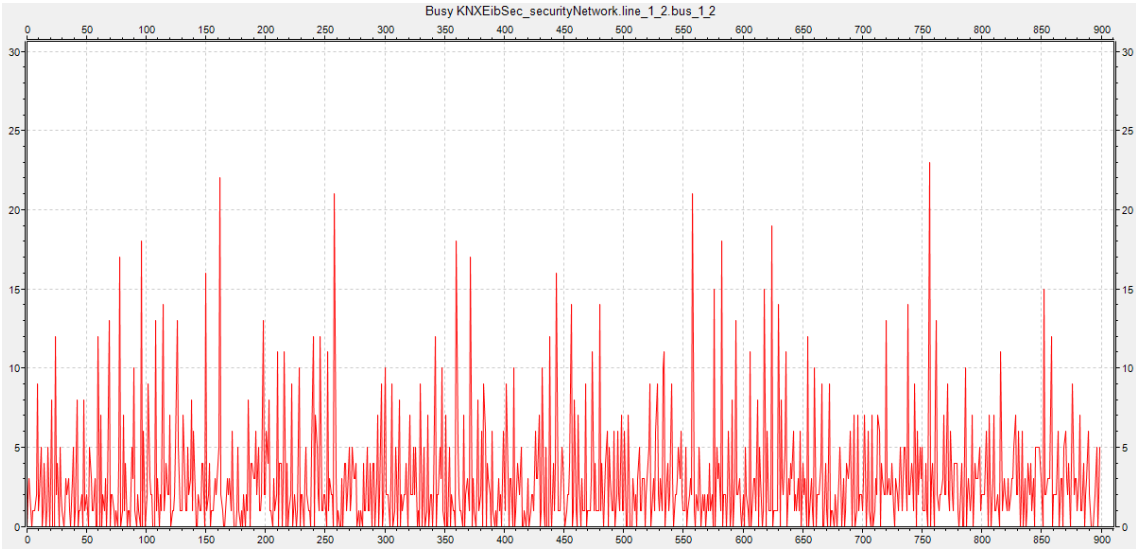


Figura 59 Respuestas Bus Ocupado línea line_1_2

6.3.3 Resumen de los resultados de simulación

Para resumir los resultados mostrados anteriormente en gráficos, mostramos tanto el máximo como su media alcanzados:

Carga de la red		Modo no Seguro	Modo Seguro
Línea 1.0	Máximo	0.244608	0.423176
	Media	0.13571214222222222	0.21668781333333334
Línea 1.1	Máximo	0.20384	0.358072
	Media	0.11872547555555556	0.18945264
Línea 1.2	Máximo	0.224224	0.358072
	Media	0.11881607111111112	0.18891010666666666

Tabla 10 Resumen medidas de carga

Un detalle apreciativo en este resumen de medidas es que tanto la línea 1.1 como la 1.2 mantienen niveles equitativos en cada modo de simulación. Esto es debido al que el tráfico cursado es el mismo en ambas líneas. El cursar el mismo número de ordenes en iguales instantes de tiempo se hecho para realizar un comparación mas real de la carga en cada unas de las líneas secundarias.

En cuanto a las respuestas del Bus Ocupado se resumen mostrando el máximo y la media de cada una de las líneas en ambas simulaciones:

Respuestas Bus Ocupado		Modo no Seguro	Modo Seguro
Línea 1.0	Máximo	27.0	28.0
	Media	2.2333333333333334	4.321111111111111
Línea 1.1	Máximo	22.0	23.0
	Media	1.69	2.9066666666666667
Línea 1.2	Máximo	21.0	23.0
	Media	1.8766666666666667	2.97

Tabla 11 Resumen medidas respuestas Bus Ocupado

Aquí un detalle importante es que en ambos modos de simulación se mantiene el mismo máximo de veces que la línea está ocupada, mientras que la media es más alta para el caso seguro. Este comportamiento es esperado ya que al haber más carga en la red el bus se mantiene ocupado más tiempo.

6.3.4 Otras medidas

Hay otras medidas comparativas calculadas en la simulación significativa para el comportamiento de cada red, no representativas gráficamente, que son:

Otras medidas		Modo no Seguro	Modo Seguro
Línea 1.0	Nº total de veces Bus ocupado	2019.0	3906.0
	Mensajes recibidos	6000.0	9906.0
	Bytes recibidos	72171.0	178308.0
Línea 1.1	Nº total de veces Bus ocupado	1530.0	2624.0
	Mensajes recibidos	6778.0	7870.0
	Bytes recibidos	61002.0	141656.0
Línea 1.2	Nº total de veces Bus ocupado	1689.0	2674.0
	Mensajes recibidos	6940.0	7905.0
	Bytes recibidos	62460.0	142286.0

Tabla 12 Resumen medidas escalares de la red

6.4 ANÁLISIS DE LOS RESULTADOS

Analizando los resultados de la simulación y comparándolos para las dos configuraciones de red simuladas en el proyecto, podemos obtener el incremento que se ha producido de la red sin seguridad habilitada a la segura. En la siguiente tabla podemos ver el porcentaje de crecimiento de una red a otra tanto en el nivel de máxima carga como en su media.

Porcentaje Incremento de carga en %		
Línea 1.0	Máximo	17,85
	Media	8,01
Línea 1.1	Máximo	15,42
	Media	7,07
Línea 1.2	Máximo	13,38
	Media	7,01

Tabla 13 Incremento carga entre los dos tipos de redes

Como podemos observar en la tabla anterior, en general tanto en el nivel máximo de carga de la línea como en la media, se obtiene un incremento de carga. Este incremento es debido al aumento de la longitud de los mensajes en la red con seguridad habilitada, ya que al realizar el cifrado nos devuelve mensajes con longitudes muchos mayores. Otra causa del incremento de carga, es el aumento del número de mensajes debido al envío previo de los mensajes de seguridad que contienen la clave y contador para el cifrado.

En el porcentaje máximo se obtiene un mayor incremento en la línea principal con respecto a las líneas secundarias. Hay un instante de tiempo en el que la línea principal tiene mayor porcentaje de carga (17,85%) con respecto al modo no seguro, causado por el retardo acumulativo de los mensajes. Los mensajes EIBSec obtendrán mayor retardo que los EIB resultando una acumulación mayor del retardo a medida que avanza el tiempo. Este retardo provoca un incremento de carga en la red causado por las retransmisiones.

En cuanto al porcentaje medio, es muy equitativo en todas las líneas. Una de las causas del incremento de la media de carga de la línea principal es el aumento de tiempo en que el bus se mantiene ocupado procesando y enviando la trama. El aumento de tiempo del bus ocupado produce un retraso en el siguiente envío y retransmisiones desde los acopladores de línea.

En cambio, el causante del incremento en media de las líneas principales no es solo el tiempo de procesamiento de los mensajes sino también los mensajes extras de seguridad procedentes del acoplador de línea recibidos en el comienzo de la simulación.

Un detalle importante a tener en cuenta es la llegada de los mensajes procedente de los dos acopladores. Dicho en otras palabras, todos los mensajes enviados de una línea a otra pasan por la línea principal. Sin embargo, las líneas secundarias procesan la trama y la envían a todos los dispositivos que tiene conectados. El resultado de estas consideraciones es una carga alta provocada por varias causas, provocando una carga mayor en una red con seguridad habilitada.

Para ver de una manera más detallada las diferencias entre los modos de simulación, en las siguientes graficas se reflejan las diferencias entre ambas en cada instante de tiempo.

Para la línea principal 1.0:

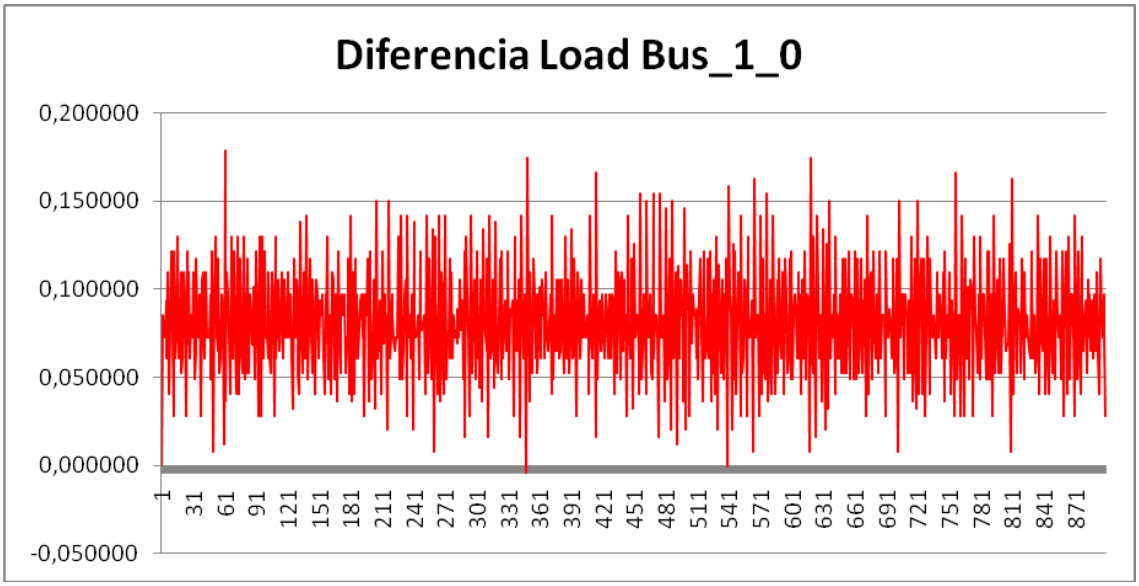


Figura 60 Diferencia carga entre modos simulados en la línea 1.0

Para la linea 1.1:

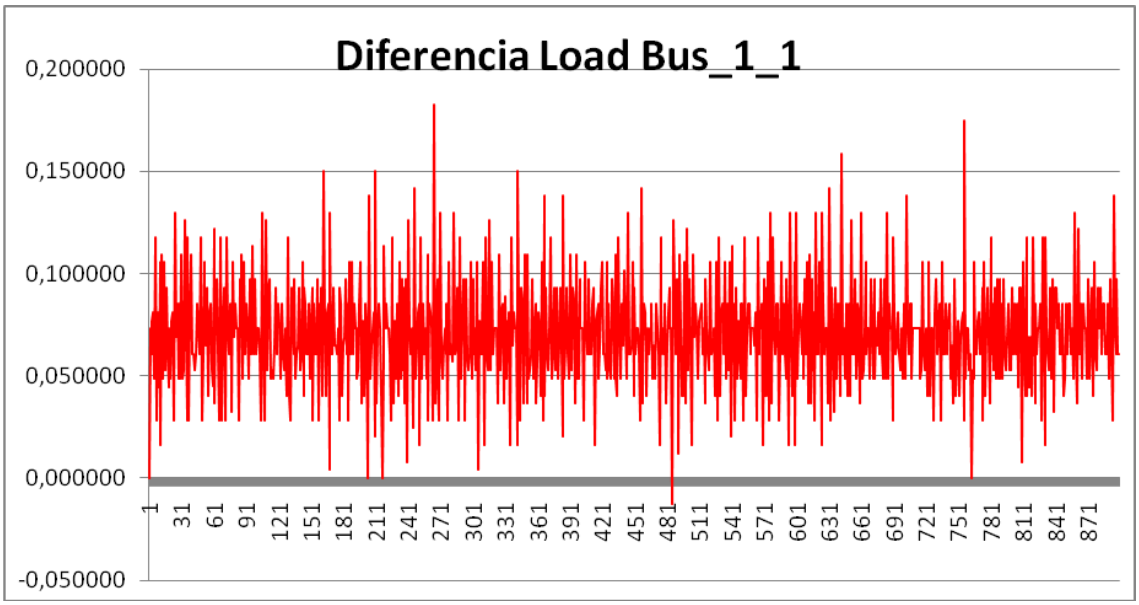


Figura 61 Diferencia carga entre modos simulados en la línea 1.1

Y para la 1.2:

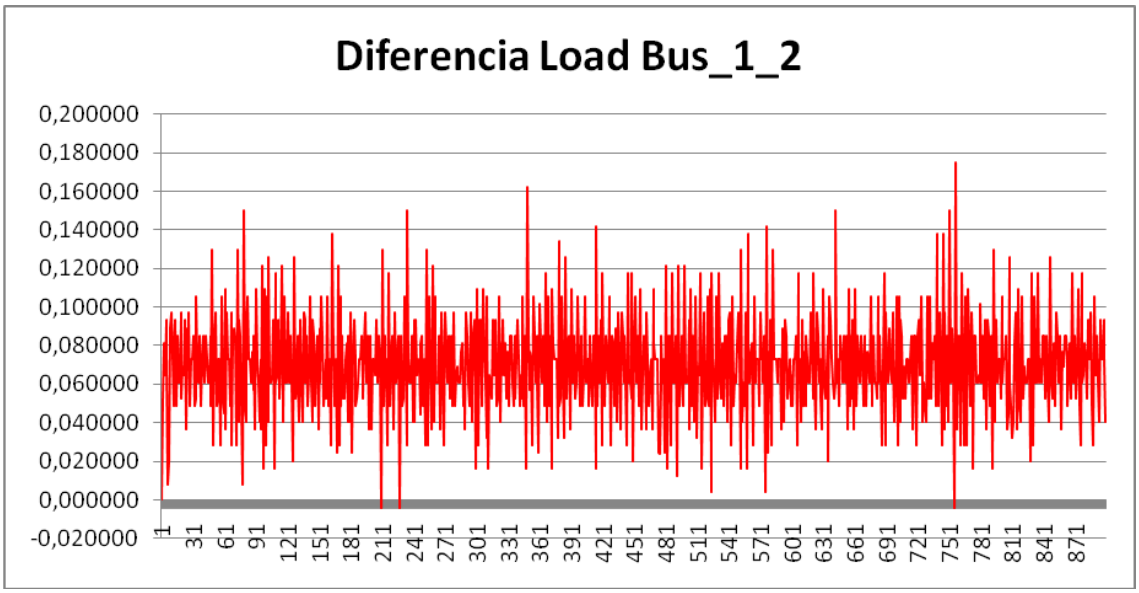


Figura 62 Diferencia carga entre modos simulados en la línea 1.2

Calculamos el porcentaje de Bus Ocupado en cada modo teniendo en cuenta el número de mensajes que se han recibido. Una vez calculado el porcentaje de ocupación de cada línea se realiza el incremento del la red no segura con respecto a la segura. Esto se refleja de manera detallada en la siguiente tabla:

	% Bus ocupado Modo no Seguro	**% Bus ocupado Modo Seguro**	**Incremento % Bus Ocupado**
Línea 1.0	33,65	39,43	5,78
Línea 1.1	22,57	33,34	10,77
Línea 1.2	24,33	33,82	9,49

Tabla 14 Porcentaje Bus ocupado

Cabe destacar que el porcentaje de ocupación de la línea principal es más alto con respecto a las líneas secundarias, ya que el tráfico enviado a la red desde los dispositivos es manejado por la línea principal.

Comparando el incremento obtenido de una red con respecto a la otra podemos observar que el incremento es mucho menor en el caso de la línea principal (aproximadamente la mitad). La obtención de menos incremento del porcentaje de colisiones en la línea principal es debido a que los mensajes de seguridad no son enviados a través de ella. Por lo tanto no ocupan tiempo de transmisión y el bus de la línea principal se mantiene menos tiempo enviando mensajes.

La conclusión es que EIBSec provoca más carga a nivel de red por la introducción de nuevos mensajes extras, aumento de la extensión de los mensajes propios del protocolo EIB y su correspondiente aumento del procesado.

Un dato a favor es la protección frente ataques externos e internos que pueda sufrir la red así como la introducción de nuevos dispositivos que inicialmente EIB no tiene desarrollados.

A la hora de la elección entre una red u otra, primero hay que constatar los requisitos necesarios de la instalación y a partir de ahí realizar la elección. Por ejemplo, si la red se va a instalar dentro de un edificio, interesaría mucho mas la protección frente a los ataques de seguridad que el nivel de carga que pueda tener la red, así como la introducción de nuevos dispositivos diseñados para ellos. En cambio, si el diseño es para una casa se podría hacer con ambas

pero interesaría una red EIB ya que el nivel de riesgo frente a ataques a la red es menor. En una casa, los dispositivos utilizados son los que inicialmente se instalaban con EIB sin descartar la posibilidad de incluir EIBSec. Otro factor influyente a tener en cuenta a la hora de instalar una tipo de red u otra es el presupuesto que resultaría elevado en el caso de instalar una red EIBSec en una casa.

El cálculo realizado hasta ahora es el porcentaje total de veces que el bus está ocupado durante la simulación. Se puede tener una visión más detallada de las diferencias existentes en cada instante de tiempo entre ambas simulaciones. Se pueden observar números negativos debido a que hay instantes que el modo seguro ha tenido menos colisiones que en el modo no seguro.

La diferencia para la línea principal 1.0 se resume con la siguiente gráfica:

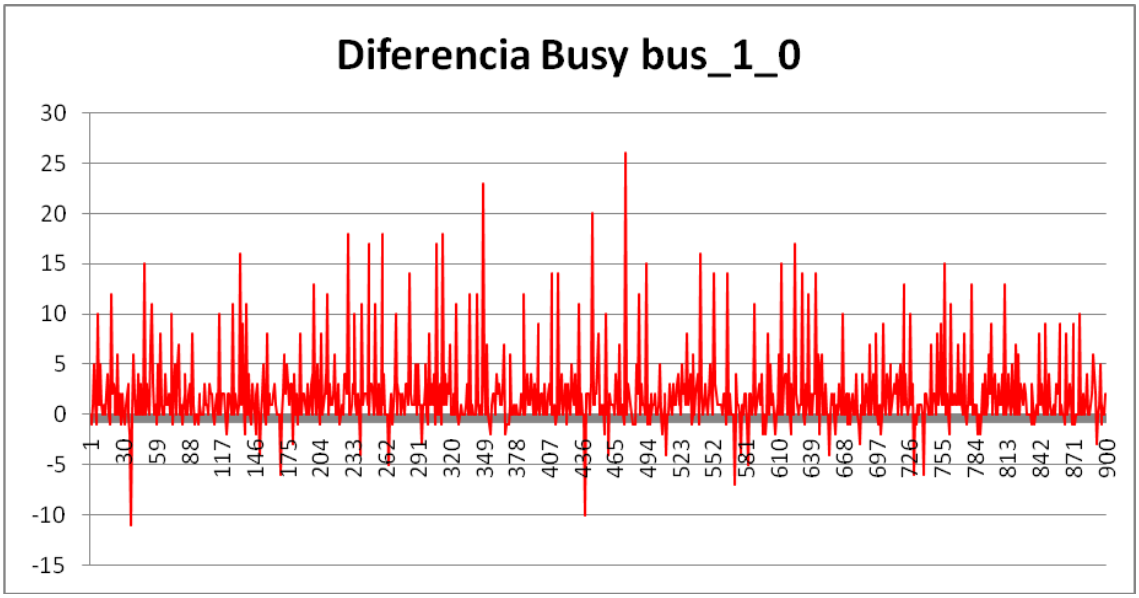


Figura 63 Diferencia Bus Ocupado entre modos en la línea 1.0

Para la linea 1.1 es:

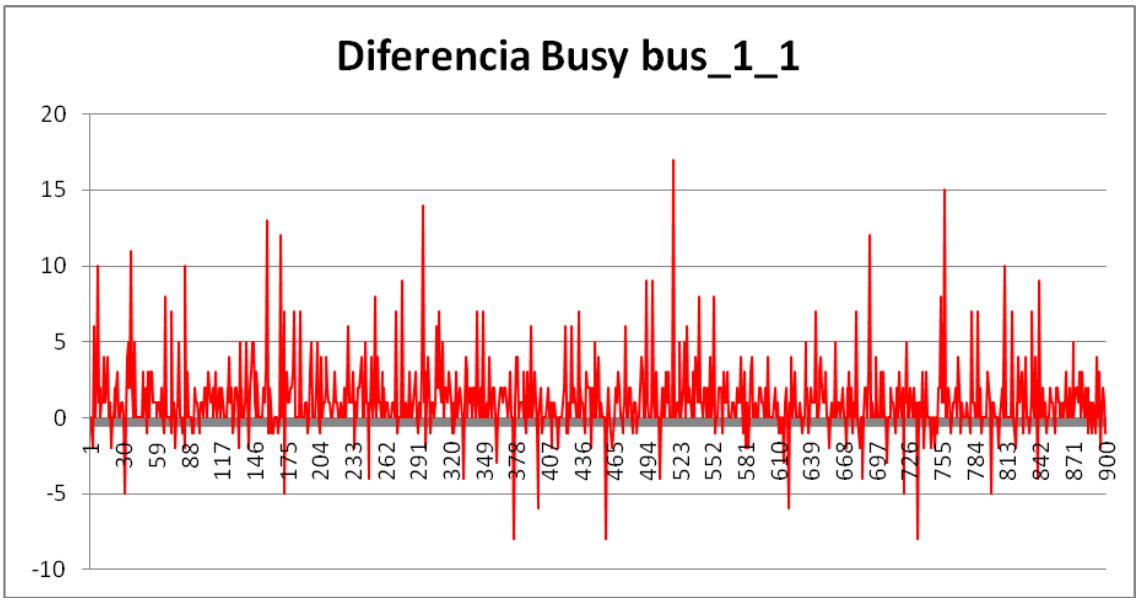


Figura 64 Diferencia Bus Ocupado entre modos simulados en la línea 1.1

Y por último la 1.2 es:

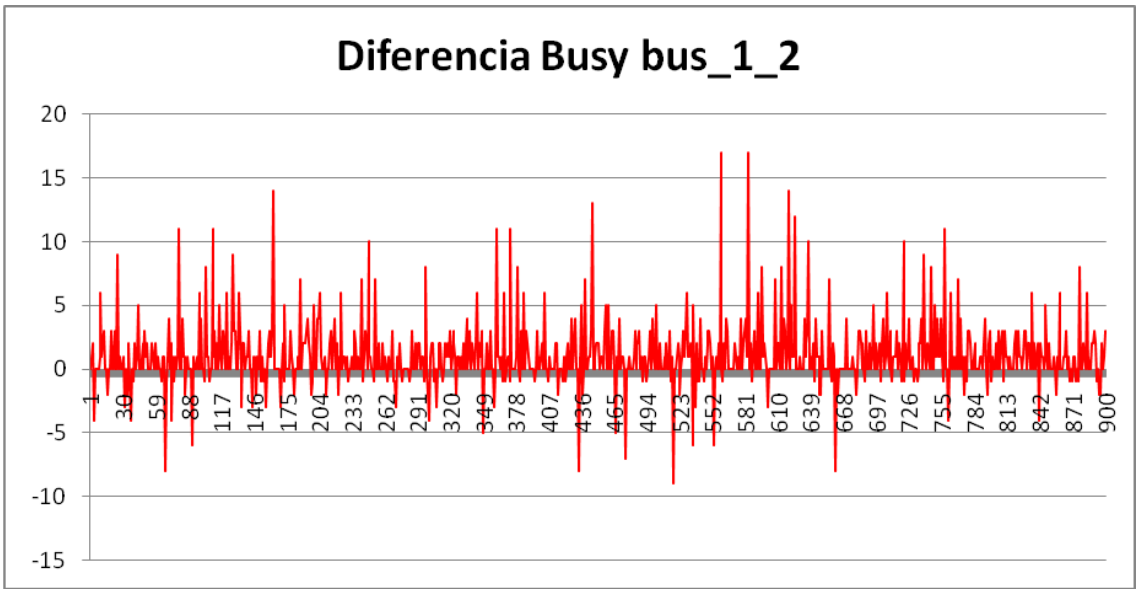


Figura 65 Diferencia Bus Ocupado entre modos simulados en la línea 1.2

Capítulo 7: HISTORIA DEL PROYECTO

En este capítulo se describen las fases en las que se ha desarrollado el proyecto con las dificultades que se hayan podido tener así como la manera de solventarlos o replanteamientos que se adoptaron para la solución desarrollada a día de hoy.

Cabe destacar en la realización del PFC que inicialmente este proyecto se consideró para ser desarrollado en lenguaje C#. En la implementación, únicamente se considero la simulación de una red domótica con el protocolo KNX/EIB. Los requisitos con el paso del tiempo han cambiado hasta llegar a lo que a día de hoy se ha desarrollado. Se ha optado por el uso de un framework llamado OMNeT++ formado por módulos simples y compuesto. El comportamiento deseado de cada uno de los componentes definidos por los módulos, es desarrollado en el lenguaje C++.

Otro cambio significativo con respecto a los requisitos iniciales es la incorporación de una variante del protocolo EIB en la que se añade seguridad en la red. La incorporación de la seguridad ha influido en la simulación final de manera que disponemos de dos redes aparentemente iguales pero con funcionalidad diferente.

Esto significó un cambio importante en el proyecto primeramente por el cambio del lenguaje de programación y como consecuencia el entorno utilizado. A este cambio se añaden el cambio de los requisitos existente y el añadido de otros nuevos. Otro detalle a tener en cuenta es el aprendizaje de la variante añadida al protocolo EIB.

Como todo cambio importante significa una manera diferente de ver las cosas y un periodo de adaptación a lo nuevo. Inicialmente, la manera de pensar fue negativa ya que por decirlo fue comenzar de nuevo. Una vez superado el periodo de adaptación tanto con el entorno como el lenguaje de programación, los inconvenientes van desapareciendo paulatinamente convirtiéndose en puntos positivos a nuestro favor.

El cambio fue a mejor en cuanto a sencillez de implementación. Un punto positivo a tener en cuenta con el cambio es el uso de un framework aparentemente sencillo. Este framework me aportó facilidad a la hora de efectuar la comunicación entre los diferentes componentes que forman la red ya que no fue necesario el uso de hilos como en el caso de C#. Considero esto una ventaja a mi favor porque la principal dificultad encontrada con el anterior lenguaje de programación fue el uso de hilos para la comunicación de la red. En

el caso de OMNeT++, como los componentes son definidos en módulos simples, heredan ciertas funciones de manejo de eventos ya desarrollados en sus librerías básicas. Actualmente la comunicación entre los componentes que forman la red EIB no ha sido el peor de los problemas ya que únicamente se desarrolla una funcionalidad diferente a partir de la ya implementado en sus librerías. Con el desarrollo de esta nueva funcionalidad definimos el comportamiento de cada módulo o componente.

En cuanto al entorno gráfico también la considero como una mejora. Es un entorno fácil de manejar en el que puedes observar tanto gráficamente el comportamiento de la red como de forma descriptiva en los logs de eventos almacenados durante la simulación.

Me gustaría decir que todo fueron ventajas pero el aprendizaje del nuevo lenguaje de programación que debía utilizar resultó más difícil de lo que esperaba. En concreto, el uso de punteros en C++ fue la parte con más dificultad.

Como he comentado anteriormente, se incorporó la extensión de seguridad de KNX/EIB. Con esta incorporación se necesitó tiempo para el estudio de los nuevos conceptos y técnicas que más tarde se debían incorporar al proyecto. Esta tarea añadió cambios en el diseño existente haciendo un nuevo diseño donde se incorporó el nuevo funcionamiento de la red.

En las nuevas tareas, cabe destacar, la introducción de imágenes de los distintos dispositivos de la red así como el cambio de estado de los actuadores durante la ejecución. Esta tarea resulto tediosa a la hora de encontrar las imágenes adecuadas y representativas que se utilizaron en entorno grafico de la simulación. También realicé tareas de procesado de imagen para encontrar el tamaño adecuado de la representación. Esta parte ha sido la más creativa y divertida de todas las tareas hechas en el proceso de desarrollo, aunque el tiempo empleado en ellas fue más de lo esperado. A la vista de los resultados gráficos obtenidos, si hago un balance entre lo tedioso y lo creativo que resultó la construcción de un entorno grafico para que fuese lo más parecido a la vida real, llego a la conclusión que ha merecido la pena.

7.1 DESCOMPOSICIÓN DE TAREAS

Para una buena planificación del trabajo inicialmente se hacen procesos de identificación y descripción de cada una de las tareas en las que se ha dividido el proyecto, así como las dependencias que puedan surgir entre ella, para el posterior desarrollo con éxito.

Para la realización del trabajo expuesto se han identificado diez tareas diferentes. A continuación se pasa a hacer una breve descripción de las tareas realizadas durante el proyecto, indicando el esfuerzo temporal que conlleva cada una de ellas.

1. *Documentación EIB.*

En esta primera fase, se realizó una lectura compresiva del funcionamiento del protocolo así como todas las características técnicas que se tienen que tener en cuenta a la hora de instalar un sistema domótico. En general, esta fase sirvió para la familiarización con el proyecto y planteamiento de su desarrollo.

Esta fase aproximadamente duró en torno a tres semanas.

2. *Instalación del entorno de desarrollo*

En esta segunda fase se realizó la instalación y adaptación del entorno de desarrollo usado durante el proyecto. Como es una herramienta libre, siempre y cuando sea para fines educativos, se opto por la descarga del paquete OMNeT para Windows. OMNeT contiene un entorno de desarrollo integrado basado en Eclipse con añadidos de editores, vistas herramientas y otras funcionalidades [10]**.**

La duración fue de una semana.

3. *Familiarización con el entorno.*

Teniendo en cuenta que dispongo de algunos conocimientos del funcionamiento de eclipse, esta fase no fue significativa. Únicamente se realizaron labores de investigación en los añadidos al entorno para su uso con OMNet++, como fueron los nuevos editores incorporados para el desarrollo y la aplicación grafica.

Su duración fue de media semana.

4. *Familiarización con el lenguaje C++*

En esta fase, se realizó un aprendizaje detallado del lenguaje de programación C++ haciendo uso de manuales básico, tutoriales y libros. No solo basto con conocer las expresiones más comunes y su funcionamiento, sino también se hizo uso de la funciones de conversión y punteros. Esta ultima parte fue lo más difícil encontrado durante el desarrollo ya que son conceptos nuevos y de difícil compresión. Este aprendizaje se hizo de forma continuada durante el desarrollo de la aplicación, a medida que se

presentaban nuevos objetivos de la funcionalidad, resultando más llevadero gracias al conocimiento de otros lenguajes de programación muy similares.

La duración fue de dos semanas.

5. *Familiarización con el framework.*

El framework OMNeT++ era completamente desconocido, por lo que se tuvo que realizar un aprendizaje completo de sus características, limitaciones, formas de uso, etc. Gracias a editores gráficos y a las completas guías de uso así como de ejemplos, el desarrollo fue más intuitivo y llevadero.

Esta fase duró dos semanas y media.

6. *Diseño de la aplicación*

Una vez familiarizado con el entorno, el lenguaje y el framework en que se desarrollaría la aplicación se procedió al diseño de la misma.

Aquí se engloban, la definición de los requisitos y las funcionalidades de la aplicación, la estructuración de los distintos módulos que componen la aplicación, la interacción entre ellos y como debían relacionarse entre sí.

La duración de esta fase fue de dos semanas.

7. *Desarrollo de la aplicación.*

En esta fase se realizó el desarrollo de todo lo diseñado en la fase anterior, con problemas que fueron surgiendo y pequeños cambios con respecto al diseño inicial. Se crearon inicialmente unas redes que servían para probar el funcionamiento del protocolo y comportamiento de los dispositivos. Mas tarde se hizo el diseño final de la redes que se utilizaron para tarde en la fase de las simulaciones. Se desarrollaron los módulos básicos, cada uno con su funcionalidad, y un fichero de texto que simula las ordenes por parte del usuario e interacciona con la red.

Después se incluyó la parte de seguridad al protocolo, reutilizando algunos módulos cuyo comportamiento es idéntico pero desarrollando otros nuevos y específicos para el modo seguro con distinta funcionalidad.

Esta fase fue la más tediosa y costosa, con una duración aproximada de unas once semanas.

8. Personalización de la aplicación

Una vez desarrollados todos los módulos que componen el proyecto, el siguiente paso a realizar fue la personalización de la interfaz gráfica de usuario. Se cambiaron las imágenes de los componentes, se añadieron mensajes de estado que cambiasen en tiempo de ejecución, se implementaron mensajes de salida gráficos y otra pequeñas mejoras en la impresión de los eventos que se guardan en el fichero de log. Con estas mejoras se pretendió crear una interfaz que fuese seria, atractiva, representativa e intuitiva para el usuario así como lo más real posible.

A la vez, se añadieron el cálculo y almacenamiento de medidas estadísticas en la red que nos ayudaron mas tarde para comparar y analizar las distintas redes.

La duración fue de una semana.

9. Simulaciones

Inicialmente esta fase fue desarrollándose a medida que se iba implementando la funcionalidad para realizar el testeo. Cuando se finalizo la implementación, se hizo un nuevo diseño de la red que fue el definitivo a utilizar en la simulación del proyecto.

Se hicieron diversas pruebas con la red ya implementada para ambas simulaciones haciendo cambios en las órdenes a ejecutar por el usuario en el fichero de texto. Finalmente, se escogió una de las pruebas para plasmar los resultados en la presente memoria, que sirviesen para analizar y comparar los resultados entre ambas. Paralelamente, se escogieron unas órdenes básicas que sirven para hacer una simulación-demo del proyecto.

Todas estas pruebas nos han servido no solo para hacer un análisis de resultados sino también para corregir errores que no se encontraron en la fase anterior.

La duración fue de unas tres semanas.

10. Documentación de la memoria

En esta última fase se ha dedicado a la documentación de la presente memoria. Ciertas partes fueron escritas a la par que el desarrollo y en otras se realizaban ciertos apuntes a tener en cuenta en el momento de documentar.

La duración fue de unas cinco semanas.

7.2 RESUMEN DE TAREAS

A continuación se incluye una tabla resumen de las tareas descritas anteriormente con un cómputo de la duración total del Proyecto realizado:

TAREA	DURACIÓN (semanas)
1: Documentación EIB	3
2: Instalación entorno de desarrollo	1
2: Familiarización con el entorno	0.5
4: Familiarización con el lenguaje C++	2
5: Familiarización con el framework	2.5
6: Diseño de la aplicación	2
7: Desarrollo de la aplicación	11
8: Personalización de la aplicación	1
9: Simulaciones	3
10: Documentación de la memoria	5
TOTAL	**31**

Tabla 15 Resumen de las tareas del Proyecto.

Por tanto, se han empleado 31 semanas en la realización del la simulación presentada.

7.3 PRESUPUESTO

A continuación, se detallan los costes que conlleva la realización del presente Proyecto. Los costes se dividen en dos bloques: los costes de materiales y los costes de personal.

Los materiales empleados son los componentes utilizados para conseguir el proyecto. Mientras que los costes de personas, son los honorarios de las personas que han participado así como el tiempo que han invertido en el trabajo. Finalmente se incluyen los costes totales del proyecto, que son las suma de los dos anteriores.

7.3.1 Costes materiales

Los costes mostrados a continuación son los asociados al material necesario para la realización del Proyecto.

CONCEPTO	PRECIO	COSTE AMORTIZADO
Ordenador Portátil	1300	148.1
TOTAL	**1300**	**148.1**

Tabla 16 Costes materiales del proyecto

El coste de amortización es calculado en base a un periodo de depreciación de 5 años.

7.3.2 Coste de personal

Aquí se recogen los costes asociados con los honorarios de las personas que han participado en la realización del proyecto. El trabajo ha sido realizado por el Ingeniero proyectista y dirigido por el tutor del presente PFC al que denominamos Jefe de proyecto.

OCUPACION	HORAS	PRECIO POR HORAS	IMPORTE
Jefe de proyecto	40	40	1600
Ingeniero	1240	32	39.680
Total	1280		41280

Tabla 17 Costes de personal del proyecto

7.3.3 Costes Totales

En la tabla 17 se recoge el desglose del coste total del proyecto:

Concepto	Precio
Costes Materiales	148.1
Costes de Personal	41280
Subtotal	41428.1
I.V.A (18%)	7.457,058
TOTAL	**48.885,158**

Tabla 18 Costes Totales

El coste total del proyecto asciendo a *cuarenta y ocho mil ochocientos ochenta y cinco con ciento cincuenta y ocho euros*.

Capítulo 8: CONCLUSIONES Y TRABAJOS FUTUROS

Como conclusiones y aspectos positivos del presente proyecto cabe destacar inicialmente los objetivos cumplidos y las características de la simulación implementada:

- ✓ Los dispositivos y líneas KNX implementados son semejantes en cuanto a funcionalidad y comportamiento a los existentes en un escenario real. Se representan aplicaciones como interruptores, reguladores, actuadores de luz, persianas y luz regulada.

- ✓ El bus se ha implementado con un comportamiento exactamente igual al que realiza en una red real. El bus reenvía los mensajes a todos los dispositivos que tiene conectados menos el que ha enviado el mensaje.

- ✓ Los usuarios pueden crear redes EIB con cualquier topología utilizando las librerías desarrolladas para el simulador. La topología de red siempre deberá seguir las especificaciones EIB. Se pueden construir diversos escenarios de prueba que pueden ir desde una simple línea con sus dispositivos, un único área como puede ser el caso de casa domóticas o múltiples aéreas para edificios automatizados. El número de dispositivos en la línea puede llegar hasta el máximo permitido en EIB. Los acopladores de línea y de backbone han sido configurados como filtros para evitar una sobrecarga de la red, definiendo una tabla de rutas con las direcciones a filtrar.

- ✓ Las tramas KNX EIB han sido implementadas tal y como los define el EIBA dividiendo el mensaje final en caracteres y calculando su paridad para su envío. Las tramas serán enviadas a través de la red según las tablas de rutas definidas en los acopladores.

- ✓ Para el caso del modo seguro de red, será el mismo mensaje pero con ciertos campos cifrados, añadiendo la funcionalidad de cifrado en protocolo. Todos los parámetros que componen las tramas son calculados y fijados en la parte del remitente y son interpretados correctamente en la parte del receptor.

- ✓ Para medir la carga de la red, se ha implementado la monitorización de cada bus de línea en intervalos de tiempo de un segundo. Las dos medidas calculadas son el número de tramas enviadas por el bus y el

número de colisiones producidas en el bus. Ambas medidas se utilizan para calcular tanto la carga de la línea en cada intervalo de tiempo como el número de colisiones que se producen en la línea.

- ✓ Al igual que en la vida real existe un usuario encargado de interactuar con los dispositivos (switchs) de la red, se ha creado un usuario simulado. Este usuario es el encargado de mandar órdenes a la aplicación para cambiar según desee el estado de los actuadores. Estas órdenes pueden ser efectuadas una sola vez o varias veces en intervalos de tiempo.

Hay otras características de implementación que se han llevado a cabo por decisión propia a la hora de realizar el proyecto:

- ✓ A la hora de incluir tipos de dispositivos en la red se han incluido los más utilizados como ya se ha comentado anteriormente. Esta decisión se ha tomado para dotar al sistema de simulación de la mayor generalidad posible sin tener que implementar todas las posibilidades existentes. Existe diferencias en el papel que desempeña cada dispositivo dentro de la línea como son:

    - *Interruptor general:* válido tanto para luces que no sean reguladas como para persianas. Enciende y apaga estos tipos de dispositivos todos a la vez.

    - *Interruptor de luz:* válido solo para luces, interactuando con estas dependiendo al grupo al que pertenezcan. Hay tan solo uno de estos que tiene definido un grupo que enciende/apaga todas las luces a la red, mientras que los demás carecen de él.

    - *Interruptor de persiana:* válido solo para persianas, enviando las órdenes dependientes del grupo al que pertenecen. Tan solo un interruptor de persiana en cada línea tiene definido un grupo que abre/cierra todas las persianas existentes en la red simulada.

    - *Reguladores de luces:* solo válido para luz regulada. Tienen varios grupos definidos y tan solo uno, igual que en casos anteriores, es capaz de subir y o bajar la intensidad de todos las luces reguladas de la red. Son los únicos dispositivos que pueden interactuar con los actuadores de luz regulada.

- ✓ En cuanto a aspectos de simulación, se ha asignado a cada tipo de dispositivo una imagen. El uso de estas imágenes nos proporcionan un aspecto más real de la red y nos permiten saber qué tipo de dispositivo está ejecutando o recibiendo la orden. Aparte de esto, durante la simulación aparecen unos pequeños mensajes de estado de los dispositivos que ayudan a comprender mejor al usuario el comportamiento de la red.

- ✓ Con respecto a las medidas que se realizan en la red, aparte de la carga y el numero de colisiones, hay otras medidas no representativas que también nos dan una idea de cómo se ha comportado la red. Algunas de estas son los paquetes enviados/recibidos, bytes enviados/recibidos, número de veces totales que la línea está ocupada, etc.

Como ya se mencionó anteriormente, en la parte de los objetivos del proyecto, no toda la funcionalidad y detalles del protocolo EIB han sido cubiertos en la simulación al igual que no se han implementado todos los tipos de dispositivos EIB. Como trabajos futuros a incluir en otro proyecto cabe destacar los siguientes:

- ✓ Las tramas de asentimiento que indican al remitente por parte del receptor que ha recibido correctamente la trama. Esta trama no se envía, aunque se tiene en cuenta el tiempo de asentimiento a la hora de calcular el tiempo que esta el bus ocupado en el envío de la trama.

- ✓ Hay otros dispositivos que no se ha implementado su orden ni comportamiento como puede ser un sensor (de cualquier tipo), alarma, aire acondicionado, calefacción, etc. Ya que todo es simulado y un sensor depende de características externas del entorno, se puede simular igual que se ha hecho con el usuario, leyendo de un fichero que el especifiquemos las condiciones externas.

- ✓ Otro aspecto del comportamiento del protocolo que no se ha implementado es el estado de alarma de la red, en el que solo se enviaran tramas cuya prioridad sea alta.

- ✓ En la parte de seguridad del protocolo cabe destacar en mejoras la implementación del funcionamiento del ACU en modo central. Se creará un acoplador que funcione única y exclusivamente como servidor de claves. Este servidor de claves se dedicara al envío y regeneración de las claves de todos los dispositivos de la red.

- ✓ Otra mejora de aspectos de simulación, es una aplicación grafica que rellenara el fichero de texto que simula las órdenes del usuario. Esta aplicación tendría todos los dispositivos que se han definido en la red (fichero NED), las posibles órdenes a ejecutar en ese dispositivo, tiempos de simulación y el número de veces que se ejecuta la orden. Con esta mejora, el usuario no tendría que saber el formato del fichero ni los tipos de órdenes que puede ejecutar cada dispositivo.

Como conclusión general, podemos decir que una vez finalizado el proyecto se han cumplido todos los objetivos inicialmente marcados. Se han simulado los dos tipos de redes para su posterior comparación. Además de esto, se ha desarrollado una aplicación que nos permitirá en un futuro simular pruebas de redes que en la vida real sean difíciles de probar por falta de recursos y presupuesto así como la introducción de nuevos dispositivos que incluya EIB en un futuro.

BIBLIOGRAFÍA

[1] *KNX Association*. Obtenido de http://www.knx.org/es/

[2] INTERNATIONAL STANDARD SO/IEC 14543

[3] Granzer, K. N. EIBsec: A Security Extension to KNX.

[4] *Wikipedia*. (s.f.). Obtenido de http://es.wikipedia.org/wiki/Dom%C3%B3tica

[5] Docencia impartida en la Universidad Politecnica de Cartagena. Departamento de Tecnología Electrónica. TEMA 9. REDES DOMÓTICAS. BUS EIB.

[6] Curso de Supervision de procesos por ordenador http://www.instrumentacionycontrol.net/es/

[7] Engineering, B. S. EIB Installation Bus: Bus System.

[8] Engineering, B. S. EIB Installation Bus: EIB Interworking Standarts.

[9] Engineering, B. S. EIB Installation Bus: EIB Serial Data Transmision.

[10] www.omnetpp.org

[11] *Omnet++ Discrete Event Simulation System Version 4.0.* (s.f.). Obtenido de www.omnetpp.org

[12] Manual Aprenda C++ como si estuviera en primero. Campus Tecnologico de la Universidad de Navarra.

[13] Vargas, Andras**.** A QUICK OVERVIEW OF THE OMNeT++ 4.0 IDE.